



National Defence
Research and
Development Branch

Défense nationale
Bureau de recherche
et développement

DREA CR/97/458

SURVEY OF DYNAMIC NEURAL NETWORK TECHNIQUES WITH APPLICATION TO TEMPORAL PROCESSING TASKS

Maryhelen Stevenson

Center for Research in Engineering and Applied Science
University of New Brunswick
P.O.Box 440, Fredericton, N.B., Canada
E3B 5A3

CONTRACTOR REPORT

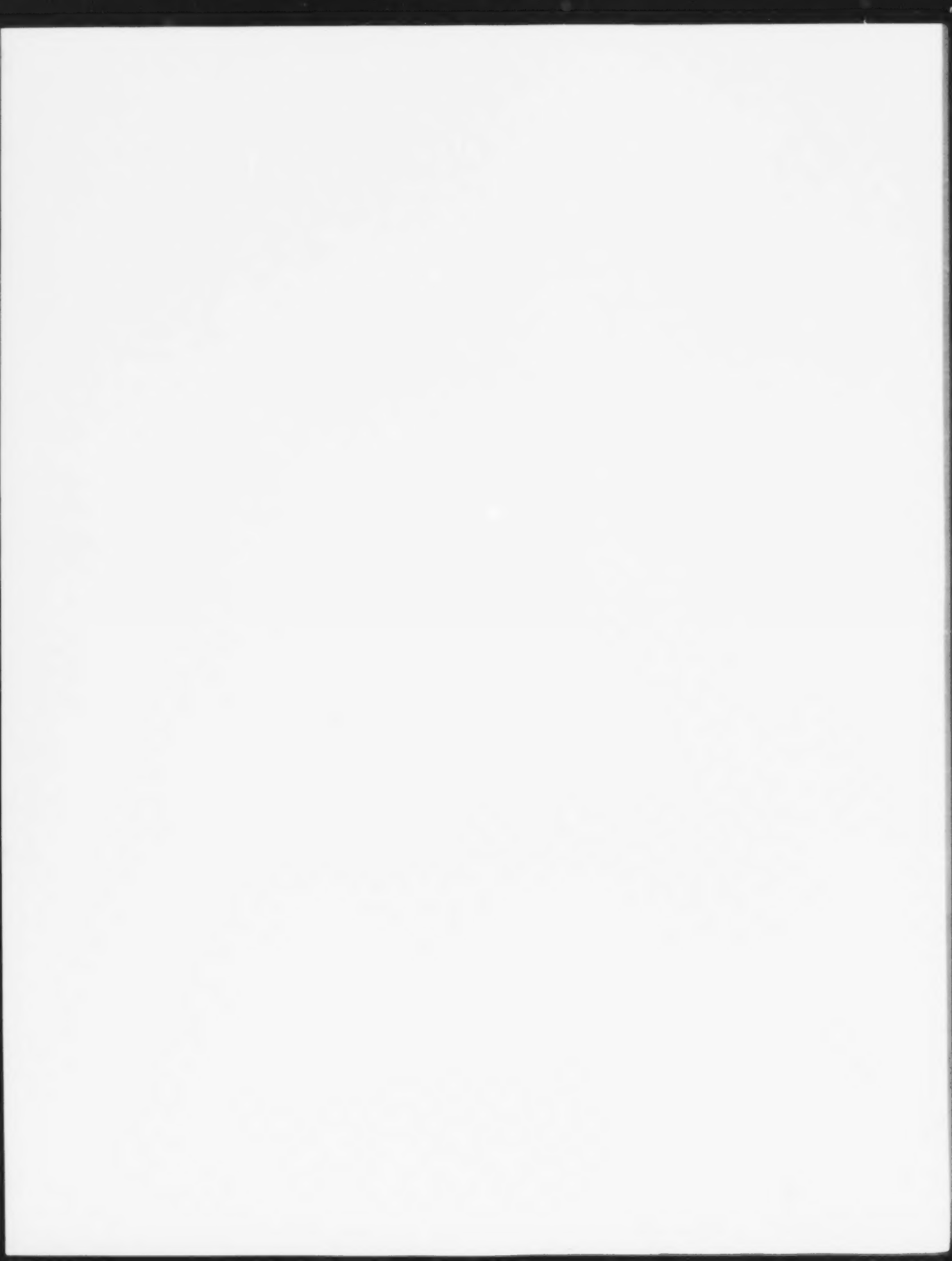
Prepared for

Defence
Research
Establishment
Atlantic



Centre de
Recherches pour la
Défense
Atlantique

Canada

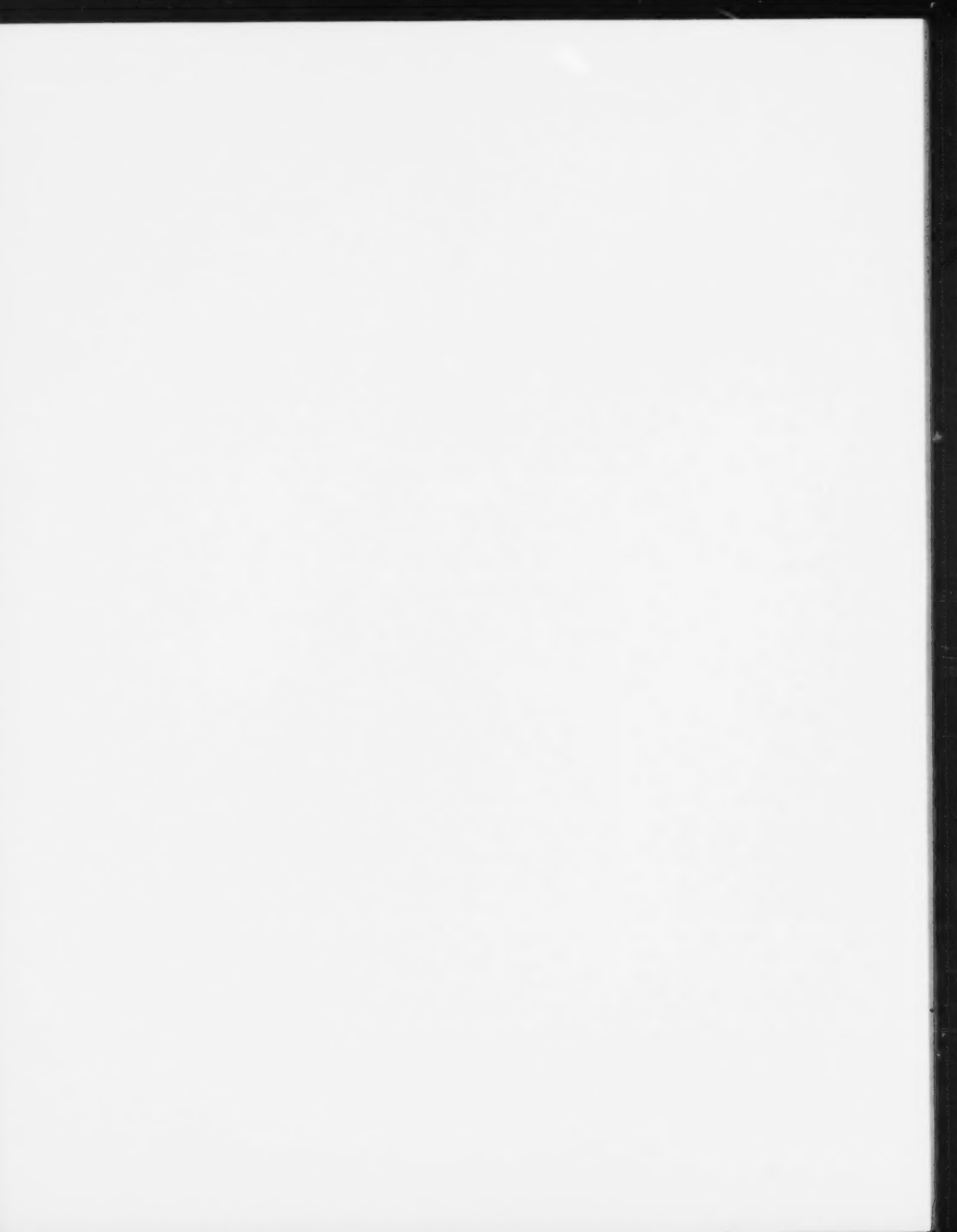


THIS IS AN UNEDITED REPORT ON SCIENTIFIC OR TECHNICAL WORK
CONTRACTED BY THE DEFENCE RESEARCH ESTABLISHMENT ATLANTIC
OF THE RESEARCH AND DEVELOPMENT BRANCH OF THE DEPARTMENT
OF NATIONAL DEFENCE, CANADA.

THE CONTENTS OF THE REPORT ARE THE RESPONSIBILITY OF THE
CONTRACTOR, AND DO NOT NECESSARILY REFLECT THE OFFICIAL
POLICIES OF THE DEPARTMENT OF NATIONAL DEFENCE.

PLEASE DIRECT ENQUIRIES TO:

THE DIRECTOR GENERAL
DEFENCE RESEARCH ESTABLISHMENT ATLANTIC
P.O. BOX 1012
DARTMOUTH, NOVA SCOTIA, CANADA
B2Y 3Z7





National Defence
Research and
Development Branch

Défense nationale
Bureau de recherche
et développement

DREA CR/97/458

SURVEY OF DYNAMIC NEURAL NETWORK TECHNIQUES WITH APPLICATION TO TEMPORAL PROCESSING TASKS

Maryhelen Stevenson

Center for Research in Engineering and Applied Science
University of New Brunswick
P.O.Box 440, Fredericton, N.B., Canada
E3B 5A3

Accepted by : 
J.N. Maksym - Contract Scientific Authority

W7707-3-2682/01-OSC
Contract Number

April 1995

CONTRACTOR REPORT

**Defence
Research
Establishment
Atlantic**



**Centre de
Recherches pour la
Défense
Atlantique**

Canada

ABSTRACT

Dynamic neural networks and their applications to temporal processing tasks are reviewed. Of special interest is a simple dynamic feedforward network known as the Finite Impulse Response Neural Network (FIRNN) and the classification of temporal patterns such as acoustic transients. Basic principles of the FIRNN are presented along with an explanation of how the FIRNN processes temporal information. Other dynamic networks which are discussed include additional dynamic feedforward neural networks, locally-recurrent globally-feedforward neural networks, partially-recurrent neural networks, and fully-recurrent neural networks. Application areas to which the various neural network paradigms have been applied include the classification of acoustic transients, prediction of chaotic time series, prediction and classification of speech signals, and modelling of nonlinear autoregressive (NAR) and nonlinear autoregressive moving-average (NAR-MA) processes.

RÉSUMÉ

Les réseaux neuronaux dynamiques et leur application à des tâches de traitement temporel sont exposés. Un réseau dynamique simple sans rétroaction, appelé réseau neuronal à réponse impulsionnelle finie (FIRNN), et la classification des profils temporels, comme ceux des transitoires acoustiques, présentent un intérêt spécial. Les principes de base du FIRNN sont présentés avec une explication de la façon dont le FIRNN traite l'information temporelle. Les autres réseaux dynamiques présentés comprennent les réseaux neuronaux dynamiques supplémentaires sans rétroaction, les réseaux neuronaux à récurrence locale et sans rétroaction globale, les réseaux neuronaux à récurrence partielle et les réseaux neuronaux à récurrence complète. Les secteurs d'application des divers paradigmes de réseau neuronal incluent la classification des transitoires acoustiques, la prévision des séries temporelles chaotiques, la prévision et la classification des signaux de parole, ainsi que la modélisation des processus autorégressifs non linéaires (NAR) et autorégressifs non linéaires à moyenne mobile (NAR-MA).

TABLE OF CONTENTS

Abstract	ii
Table of Contents	iii
1 Objective and Organization of Report	1
2 Introduction to Neural Networks	2
2.1 Fundamental concepts of neural networks	2
2.2 Applications of feed-forward neural networks to sonar signal processing ..	4
2.3 Static neural networks versus dynamic neural networks	5
3 Time Delay / Finite Impulse Response Neural Networks	7
3.1 Basic processing unit of an FIR neural network	7
3.2 Notation and description of the FIR neural network	8
3.3 Static-network implementation of an FIR neural network	10
3.4 How the FIRNN/TDNN processes temporal information	15
3.5 The temporal backpropagation training algorithm	15
3.5.1 Derivation of temporal backpropagation	17
3.5.2 Implementation of temporal backpropagation	20
3.6 Training the FIRNN as a sequence classifier: architecture, desired-response sequences, and error-scaling techniques	22
4 Successful Applications of FIR/TD NN's	25
4.1 Speech and phoneme recognition	25
4.1.1 Suggested TDNN/FIRNN architecture for time-invariant speech/phoneme classification	26
4.1.2 Application of the TDNN/FIRNN to phoneme recognition	27
4.2 Application of the FIRNN to problems in time series prediction	28
5 Review of Dynamic Neural Networks	32
5.1 Dynamic feedforward neural networks	33
5.1.1 The time-derivative neural network architecture	33
5.1.2 The tempo 2 network	35
5.2 Locally recurrent globally feedforward (LRGF) neural networks	38
5.2.1 Neural networks with IIR synapses (Back-Tsoi)	38
5.2.2 Frasconi-Gori-Soda local feedback networks	39

5.2.3 Memory neuron networks	40
5.3 Fully recurrent networks (Williams-Zipser)	42
5.4 Partially recurrent neural networks as NAR and NARMA models	43
5.5 Comparison of LRGF, static feedforward, and fully recurrent neural networks for Speech Prediction	45
5.6 Training algorithms for recurrent neural networks	46
6 Classification of Acoustic Transients Using Neural Networks	48
6.1 Description of the DARPA data set	49
6.2 Description of the DARPA data set by Ghosh, Deuser, and Beck	49
6.2.1 Preprocessing and feature extraction	50
6.2.2 Classifiers attempted and summary of classification results	51
6.2.3 Synergistic approach to classification (Ghosh et al.)	51
6.3 Description of the DARPA data set by Kundu, Chen, and Persons	52
6.3.1 Preprocessing and feature extraction	52
6.3.2 Classifiers attempted and summary of classification results	53
6.3.3 Synergistic approach to classification (Kundu et al.)	53
6.4 Description of the DARPA data set by Hemminger and Pao	54
6.4.1 Preprocessing and feature extraction	55
6.4.2 Classifiers attempted and summary of classification results	55
6.5 Application of dynamic neural networks to acoustic transient classification	56
7 Summary and Conclusions	58
REFERENCES	61

1 Objective and Organization of Report

The purpose of this report is to review dynamic neural networks and their applications to temporal processing tasks. Of special interest is a simple dynamic network known as the Finite Impulse Response Neural Network (FIRNN) and its potential applications to the classification of acoustic transients.

The organization of the report is as follows. Chapter 2 presents a brief overview of neural network basics; this overview includes background material on static neural networks and introduces the concept of dynamic neural networks. In chapter 3, the FIRNN is introduced along with an equivalent neural network paradigm known as the Time Delay Neural Network (TDNN). Included in chapter 3 are: a procedure for unfolding the FIRNN into an equivalent static network, a discussion on how the FIRNN processes temporal information, and a derivation and presentation of the Temporal Backpropagation training algorithm (along with a discussion of implementation issues). Chapter 4 highlights successful applications of the FIRNN/TDNN to temporal processing tasks. A review of dynamic neural networks and problems to which they have been applied is included in chapter 5. The general categories of dynamic neural network architectures which are discussed include: dynamic feedforward architectures, locally recurrent globally feedforward architectures, fully recurrent networks, and partially recurrent networks. Training algorithms are not discussed in detail but references are provided. Chapter 6 reviews the application of neural networks to classification of acoustic transients received by a passive sonar system. The report concludes with chapter 7.

2 Introduction to Neural Networks

The field of neural networks has experienced tremendous growth over the past decade. In the mid 1980's, the field was rejuvenated due, in part, to advances in training algorithms and advances in VLSI technology. Since the mid 1980's, interest in neural networks has continued to grow; much of the continued growth is a result of the successful application of neural networks to difficult problems spanning a wide variety of disciplines [54].

Adding to the appeal of neural networks are the following facts:

- Neural networks learn by example (*i.e.*, they have the ability to learn a desired mapping from a representative sample of the mapping). This is an attractive characteristic since it is often much easier to supply a training set of patterns and their associated desired responses than it is to generate a set of rules which mechanizes the procedure for mapping each pattern into its desired response.
- Neural networks are adaptable. This means that if the environment, to which the neural network is applied, changes slowly with time, the parameters of the network can be adapted over time so as to maintain optimal performance.
- Neural networks are universal approximators [24,7,18]. This is an important capability since solutions to many real-world problems require a processor capable of implementing complicated nonlinear relationships.
- Neural networks are collections of processors operating in parallel. This means that they offer the advantage of high-speed processing¹.

2.1 Fundamental concepts of neural networks

A neural network consists of many interconnected nonlinear processing units. Each processing unit maps many inputs into a single output. A typical processing unit, as suggested by McCulloch and Pitts [31] in 1943, is shown in Figure 1. As can be seen from the figure, each input (including the bias input which is fixed to a value of +1) is scaled by an associated adjustable weight. The output of the processing unit is a nonlinear function, $f(\cdot)$, of the weighted sum of inputs, s . By adjusting the values of the weights, one changes the input/output map of the processing unit.

There are several different nonlinear functions which are used in practice [29,17]. The most commonly used belong to the class of sigmoid (s-shaped) functions. Sigmoid functions are both differentiable and monotonic which means that gradient descent techniques

1. It should be noted that many neural network are simulated on digital computers using a single processor and therefore do not take advantage of the parallel processing capability. This is mainly due to the abundance of digital computers and the relative scarcity of dedicated neural hardware.

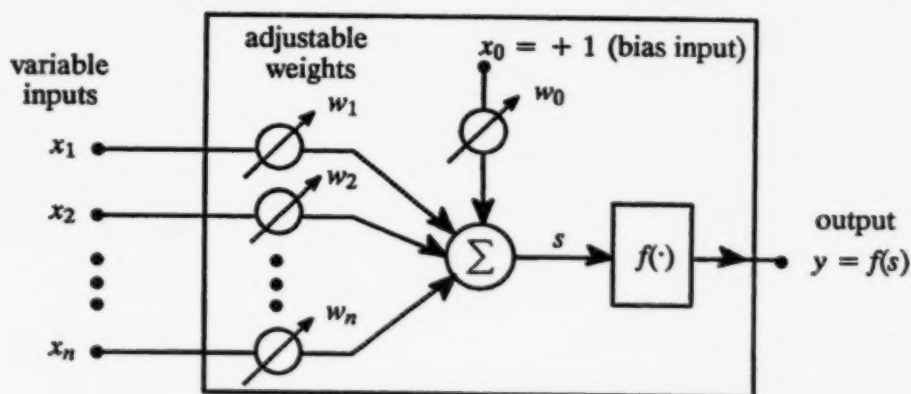


Figure 1: Basic processing unit of a static neural network.

(such as backpropagation [41]) can be applied to adjust the weights of the network so as to minimize a selected cost function (typically, the cost function is chosen to be the average squared difference between the desired network output and the actual network output). An example of a sigmoid function is the hyperbolic tangent which is plotted in Figure 2. Note that the hyperbolic tangent ranges between -1 and 1 as opposed to the logistic sigmoid which ranges between 0 and 1 . In many cases, networks which employ the hyperbolic tangent function tend to learn faster than networks which employ the logistic sigmoid [17].

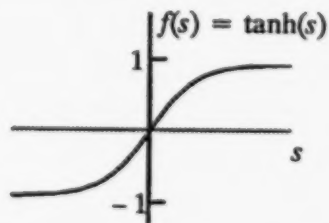


Figure 2: The hyperbolic tangent function

Inputs to processing units can originate either from sources external to the network or from the outputs of units within the network. Often, the processing units are arranged in layers and the outputs of one layer serve as inputs to the units of the following layer (see Figure 3). Such a network is termed a *layered feed-forward neural network*. In the case of a layered feed-forward neural network, the external inputs are the inputs to the processing

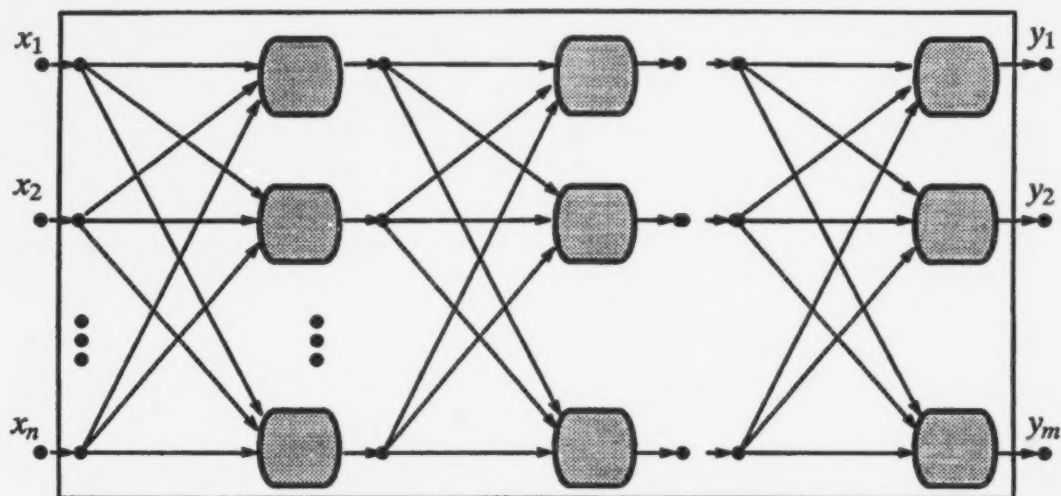


Figure 3: A layered feed-forward Neural Network

units of the first layer; the external outputs of the network are the outputs of the processing units in the last layer.

2.2 Applications of feed-forward neural networks to sonar signal processing

Among the plethora of neural network applications which have been reported in the last few years are several which involve the processing of sonar signals. Within the neural network community, the best-known of these is perhaps a study conducted by Gorman and Sejnowski [15, 16] in which a layered feed-forward neural network was trained to classify (according to target-type) the target echoes² from an active sonar system. Once trained, the neural network was able to correctly classify close to 100% of the returns in the training set and close to 90% of the returns in the test set. For comparison, human listeners were trained on the same returns³ included in the neural network's training set. The average performance of the trained⁴ human listeners was 91% on the training set (approximately 9% worse than the neural network); their performance on the test set had not

2. The sonar returns were preprocessed prior to being used as input to the neural network. The result of the preprocessing was a spectral envelope consisting of 60 points (each point of the spectral envelope represented the amount of energy in the associated frequency bin over the duration of the return).

3. The human listeners were presented with the raw signal data shifted down to auditory frequencies.

4. Training of the human listeners was terminated only after it was clear that their performance would not improve with further training.

been evaluated at the time the study was published. The performance of the neural network in this study was very encouraging in that sonar target recognition is an example of a task which is difficult to automate and for which it is difficult to achieve human-like performance.

Additional work using neural networks to classify signals received by a passive sonar system is discussed in chapter 6.

2.3 Static neural networks versus dynamic neural networks

The study by Gorman and Sejnowski used a feedforward neural network trained by the backpropagation algorithm. The map realized by this network was *static* in the sense that the classification decision (*i.e.*, the output of the network) depended only the present set of network inputs; in this case, the inputs consisted of a set of features representing the spectral content of the returned signal over its duration. In many cases, analysis of data can be greatly enhanced by exploiting the dynamic evolution of the data. In order to do this, the preprocessing of the data should be such that it produces a series of feature sets representing a series of short time windows of the data. Once this is done, there are several ways to proceed. The simplest and most commonly-used method is to concatenate the series of feature sets so as to produce one big feature set which is then applied to a standard feed-forward neural network. In this case, we have represented time explicitly as an additional spatial dimension to the input. Elman [11] and Mozer [32] note that there are several problems which accompany this approach of "parallelizing time"; for instance, this approach:

1. requires an interface to the outside world which buffers the input
2. imposes a rigid limitation on the duration of patterns, and
3. "does not easily distinguish relative temporal position from absolute temporal position" as a temporally shifted version of a pattern is often quite spatially distant from the original pattern.

Indeed, if the key to correct analysis of the data is to check for a particular sequence of short-duration events, the neural network may have a hard time finding the correct solution. In a sense, we have withheld information (or hints) from the network by presenting the series of feature sets simultaneously and not advising the network that correct classification may depend on the order in which events occur. One way to provide the network

with this information is to choose a network architecture which allows time to be represented implicitly rather than explicitly.

Throughout this report, we will differentiate between a static neural network and a dynamic neural network as follows. A *static* neural network is one whose output depends only on the current set of inputs, whereas the output of a *dynamic* neural network may depend on past inputs and/or past outputs as well as the current set of inputs. In short, a dynamic network has memory.

There are several methods by which one can modify a standard static feedforward neural network so as to yield a dynamic neural network. One of the simplest methods is to modify each processing unit of the neural network so that its output depends on both present and past values of its inputs; this can be easily done by replacing each existing input connection by a finite impulse response (FIR) filter (a tapped delay line of connections, one connection for each tap). The resulting architecture has been referred to both as a time-delay neural network (TDNN) [25,45] as well as an FIR neural network (FIRNN) [47]. A second method is to feed back past outputs (or processed versions of past outputs) of the network for use as additional inputs to the neural network. Other methods include replacing existing input connections to processing units by infinite impulse response (IIR) filters, and using past outputs of individual processing units as additional inputs to the processing units.

A detailed discussion of the FIRNN/TDNN is provided in chapter 3. A review of other dynamic neural networks and problems to which they have been applied is included in chapter 5.

3 Time Delay / Finite Impulse Response Neural Networks

As previously noted, the TDNN/FIRNN architecture results when each connecting weight of a feed-forward static network is replaced by a tapped delay line. The operation of the TDNN is exactly the same as that of the FIRNN. The only difference between the two paradigms is the description/implementation of the training algorithms. It should also be noted that although the two training procedures differ in implementation, they are equivalent in that the resulting weight updates are identical.

As described by Waibel *et al.* [45], the TDNN is trained by unfolding the network in time (the unfolding procedure is described in section 3.3), constraining certain groups of weights in the unfolded network to have the same values, and applying standard backpropagation [41] to the unfolded network. Each weight in a group of weights which are constrained to have the same value is updated by an amount equal to the average of the individually prescribed weight updates.

The FIR neural network is trained via a process, introduced by Wan [47,48,49], termed *temporal backpropagation*. Although temporal backpropagation results in the same weight updates as the procedure described by Waibel *et al.*, it is more efficient and does not require the replication of any units in the network. It is for this reason that we have chosen to implement temporal backpropagation for the experimental work of this contract. The details of temporal backpropagation will be discussed in section 3.5.

Throughout the remainder of this report, we will use the terms FIR neural network and time delay neural network interchangeably.

3.1 Basic processing unit of an FIR neural network

The basic processing unit of an FIR neural network is shown in Figure 4. Note that the scalar connections of the static processing unit shown in Figure 1 have now been replaced by Finite Impulse Response filters [53] (the detail of which is also shown in the figure). For convenience in distinguishing the FIR connecting filters of order M from the FIR processing units with order M connections, we have labelled the connecting filters as **fir:M** and the processing units as **FIR:M**. The order of an FIR connecting filter is defined to be number of delay elements employed by the filter; each delay element is denoted by

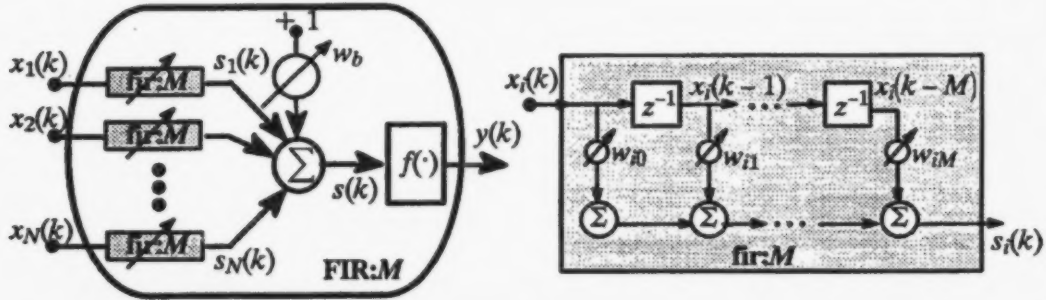


Figure 4: Basic processing unit of an FIR neural network (left) and a detailed sketch of an FIR synapse of order M .

the unit delay operator, z^{-1} . Thus, the output, $y(k)$, of an FIR: M processing unit is a function of the past M sets of inputs as well as the present set of inputs.

3.2 Notation and description of the FIR neural network

An FIR neural network consists of many FIR processing units arranged in the feed-forward layered fashion of Figure 3. For convenience in describing the architecture of an FIR neural network, we introduce the following notation. We use L to denote the number of layers of processing units in the network. We use $N^{(0)}$ to denote the number of external inputs to the network and $N^{(\ell)}$ to denote the number of processing units in layer ℓ of the network. We assume that all FIR filters connecting the nodes on layer $\ell - 1$ to the nodes on layer ℓ are of the same order which we denote by $M^{(\ell)}$. Using the notation introduced above, we can concisely describe the architecture of an FIR network as:

$$\text{FIR} \left[\left(N^{(0)} : 0 \right) \times \left(N^{(1)} : M^{(1)} \right) \times \dots \times \left(N^{(L)} : M^{(L)} \right) \right].$$

As an example, an FIR[(1:0) x (2:1) x (2:2) x (1:3)] is illustrated in Figure 5. Note that this network has one input, two nodes on the first layer, two nodes on the second layer and one node on the third (or output) layer. The filter orders for the first, second, and third layers are 1, 2, and 3 respectively.

We also introduce the following notation (illustrated in Figure 6 which shows a detailed sketch of the output node for the network of Figure 5) to describe the various weights and signals of the network. The vector of weights associated with the FIR filter connecting

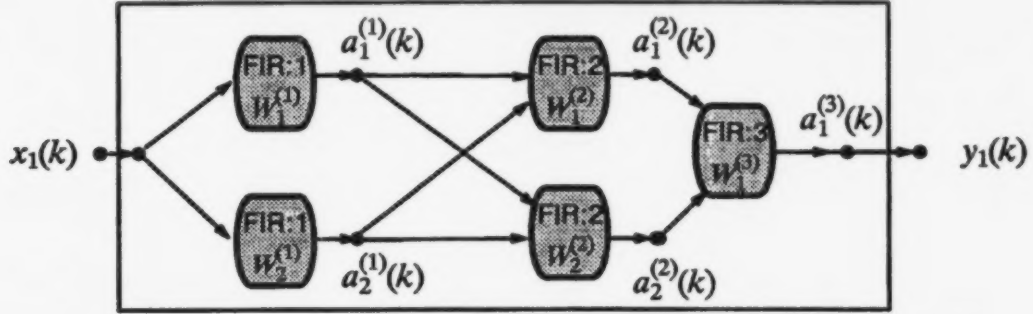


Figure 5: An FIR network with architecture FIR[(1:0) x (2:1) x (2:2) x (1:3)].

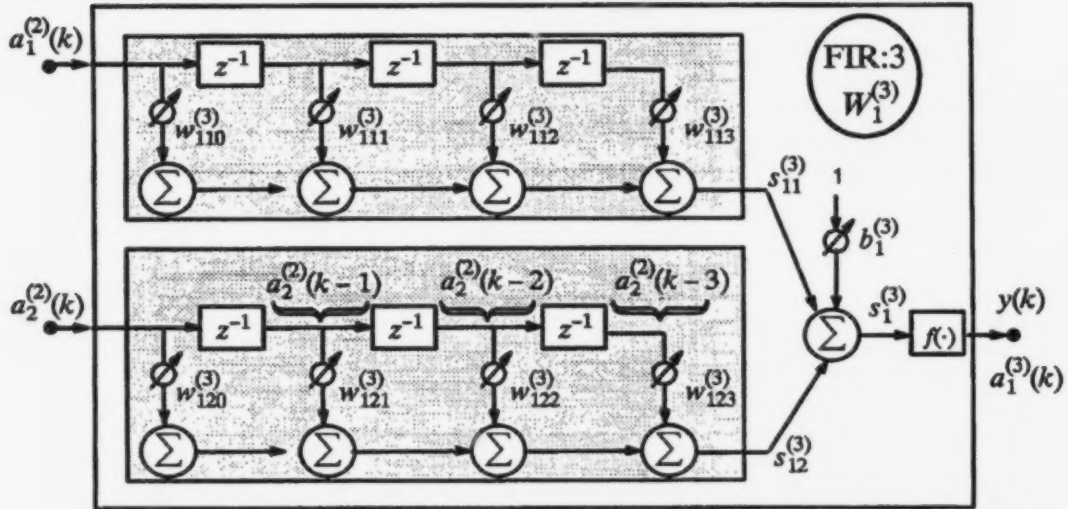


Figure 6: A detailed sketch of the output node for the FIR network of Figure 5.

node m in layer ℓ to node q in layer $\ell + 1$ is denoted as $w_{qm}^{(\ell+1)}$. The individual weights of this filter are denoted by $w_{qmn}^{(\ell+1)}$ for $0 \leq n \leq M^{(\ell+1)}$. Thus, $w_{qmn}^{(\ell+1)}$ is the weight which scales $a_m^{(\ell)}(k-n)$ at time k , where $a_m^{(\ell)}(k)$ is the output of node m in layer ℓ at time k . The output of the FIR filter connecting node m of layer ℓ to node q of layer $\ell + 1$ at time k is denoted by $s_{qm}^{(\ell+1)}(k)$. Letting $\mathbf{a}_m^{(\ell)}(k)$ denote the vector of the current and $M^{(\ell+1)}$

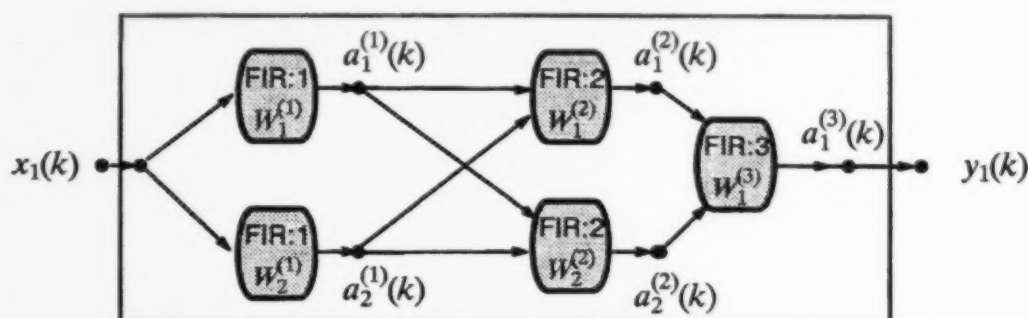


Figure 5: An FIR network with architecture $\text{FIR}[(1:0) \times (2:1) \times (2:2) \times (1:3)]$.

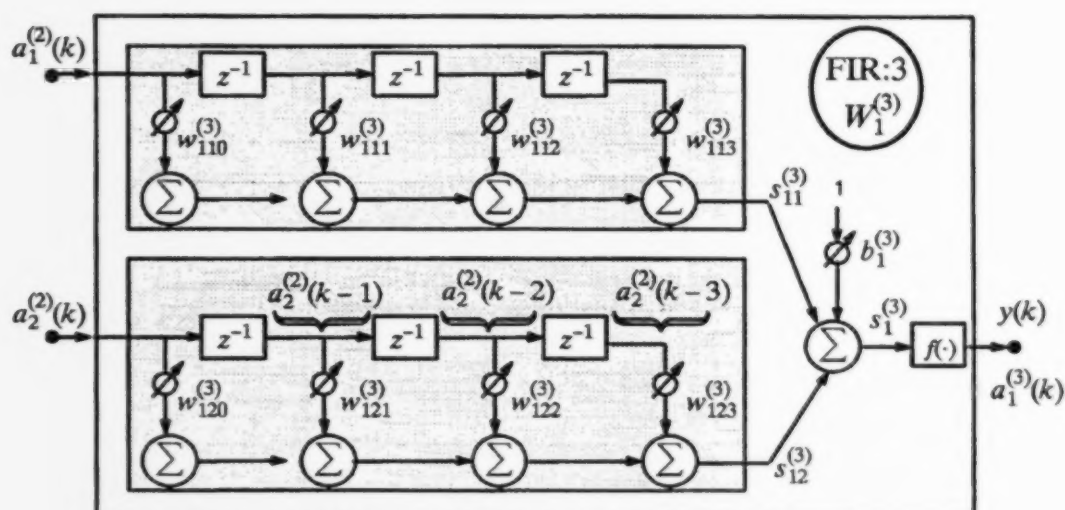


Figure 6: A detailed sketch of the output node for the FIR network of Figure 5.

node m in layer ℓ to node q in layer $\ell + 1$ is denoted as $w_{qm}^{(\ell+1)}$. The individual weights of this filter are denoted by $w_{qmn}^{(\ell+1)}$ for $0 \leq n \leq M^{(\ell+1)}$. Thus, $w_{qmn}^{(\ell+1)}$ is the weight which scales $a_m^{(\ell)}(k-n)$ at time k , where $a_m^{(\ell)}(k)$ is the output of node m in layer ℓ at time k . The output of the FIR filter connecting node m of layer ℓ to node q of layer $\ell + 1$ at time k is denoted by $s_{qm}^{(\ell+1)}(k)$. Letting $\mathbf{a}_m^{(\ell)}(k)$ denote the vector of the current and $M^{(\ell+1)}$

past values of $a_m^{(\ell)}(k)$, we can write that:

$$s_{qm}^{(\ell+1)}(k) = \sum_{n=0}^{M^{(\ell+1)}} a_m^{(\ell)}(k-n) w_{qmn}^{(\ell+1)} = \left(\mathbf{w}_{qm}^{(\ell+1)} \right)^T \mathbf{a}_m^{(\ell)}.$$

The *linear output* of node q in layer $\ell + 1$ at time k , denoted by $s_q^{(\ell+1)}(k)$, is simply the sum of the associated bias weight and the connecting filter outputs at time k :

$$s_q^{(\ell+1)}(k) = b_q^{(\ell+1)} + \sum_{m=1}^{N^{(\ell)}} s_{qm}^{(\ell+1)}(k)$$

while the *nonlinear output* of this node, denoted by $a_q^{(\ell+1)}(k)$, is a nonlinear function $f(\cdot)$ of the node's linear output:

$$a_q^{(\ell+1)}(k) = f\left(s_q^{(\ell+1)}(k)\right).$$

For convenience, we introduce a special notation for the external inputs and outputs of the neural network. We let the inputs to the network at time k be denoted by $\mathbf{x}(k)$ and the outputs be denoted by $\mathbf{y}(k)$, where:

$$\mathbf{x}(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_{n_0}(k) \end{bmatrix} = \begin{bmatrix} a_1^{(0)}(k) \\ a_2^{(0)}(k) \\ \vdots \\ a_{n_0}^{(0)}(k) \end{bmatrix} \quad \text{and} \quad \mathbf{y}(k) = \begin{bmatrix} y_1(k) \\ y_2(k) \\ \vdots \\ y_{n_L}(k) \end{bmatrix} = \begin{bmatrix} a_1^{(L)}(k) \\ a_2^{(L)}(k) \\ \vdots \\ a_{n_L}^{(L)}(k) \end{bmatrix}.$$

3.3 Static-network implementation of an FIR neural network

Although the processing unit of Figure 4 is dynamic in the sense that it has memory (present outputs depend on past inputs), it is easy to see that by spatially encoding the temporal information (of the input sequences, we can accomplish the same map from input sequences to output sequence with a static processing unit. A static processing unit, equivalent to the FIR processing unit of Figure 4, is shown in Figure 7. Note that the equivalent static processing unit has the same number of weights as the FIR processing unit; this is not the case for a multilayer FIR neural network.

Given a multilayer FIR neural network, the easiest way to find the equivalent static network is to replace the FIR processing units by their equivalent static processing units one layer at a time, starting with layer L and working backward through the network to

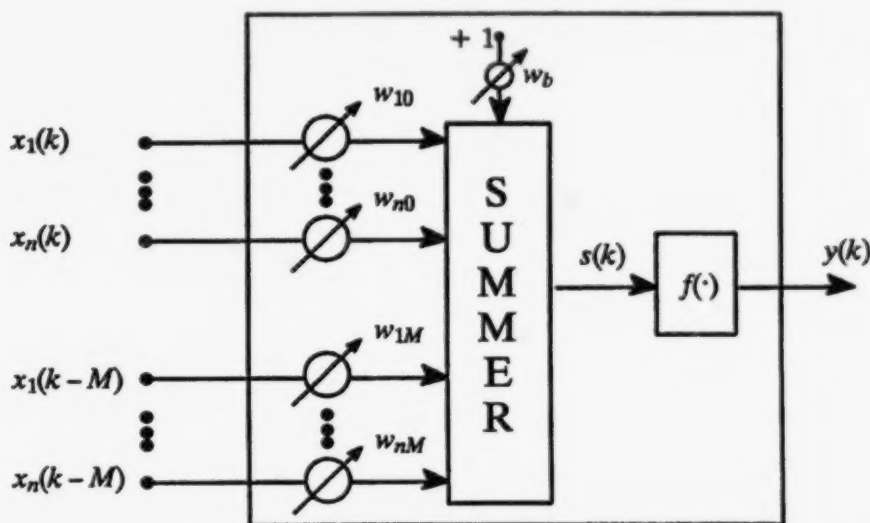


Figure 7: This static processing unit together with inputs which include spatially-encoded temporal information is equivalent to the dynamic FIR processing unit of Figure 4.

layer 1. For example, in order to find the equivalent static network for the $\text{FIR}[(1:0) \times (2:1) \times (2:2) \times (1:3)]$ network of Figure 5, we begin by replacing the output-layer node by its static equivalent.

A detailed sketch of the output node for the network of Figure 5 was shown in Figure 6 and its equivalent static replacement is shown in Figure 8. From Figure 8, we see that the equivalent static network will require, as inputs, the present and past three values of the original second-layer outputs. In order that these inputs be available simultaneously, the new second layer must include four copies of each of the original second-layer nodes; one copy will operate on the original second-layer input sequences while the other copies will operate on appropriately delayed versions of the original second-layer sequences. This is illustrated in Figure 9.

Since the order of the second-layer FIR nodes is two, we see that in order to replace each of the second-layer FIR nodes by an equivalent static node, the first-layer units will need to provide the present and past five values of the original first-layer outputs: $a_1^{(1)}(k)$ and $a_2^{(1)}(k)$. Thus, the first-layer of the equivalent static network will have six times as

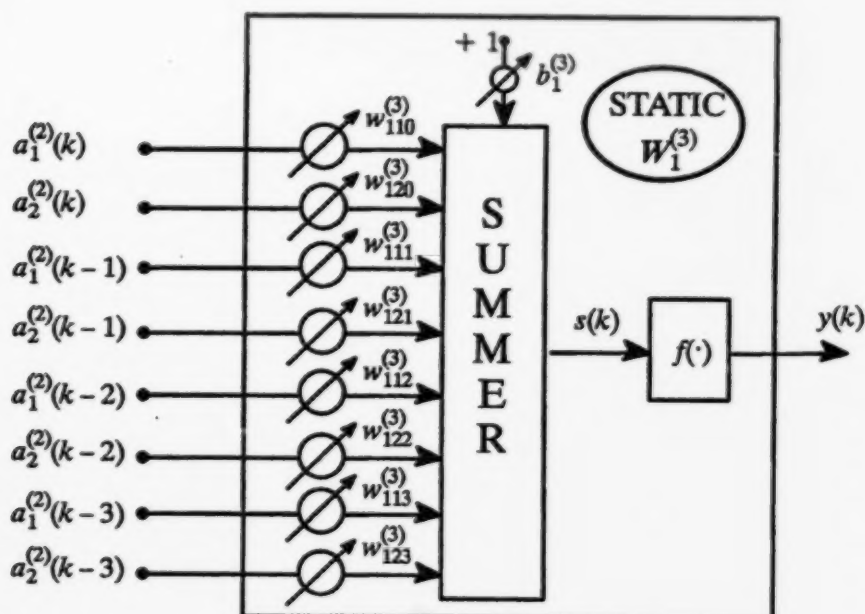


Figure 8: Static equivalent of the output node shown in Figure 6.

many nodes as the first-layer of the original FIR network. The equivalent static network is shown in Figure 10.

From the example above, it is easy to generalize this “unfolding” procedure to an arbitrary FIR network. In general, the number of nodes in the ℓ th layer of the equivalent static network will increase by a factor of $M^{(L)} + M^{(L-1)} + \dots + M^{(\ell+1)} + 1$ over that of the original FIR network. The number of inputs to the network will increase by a factor of $M^{(L)} + M^{(L-1)} + \dots + M^{(1)} + 1$.

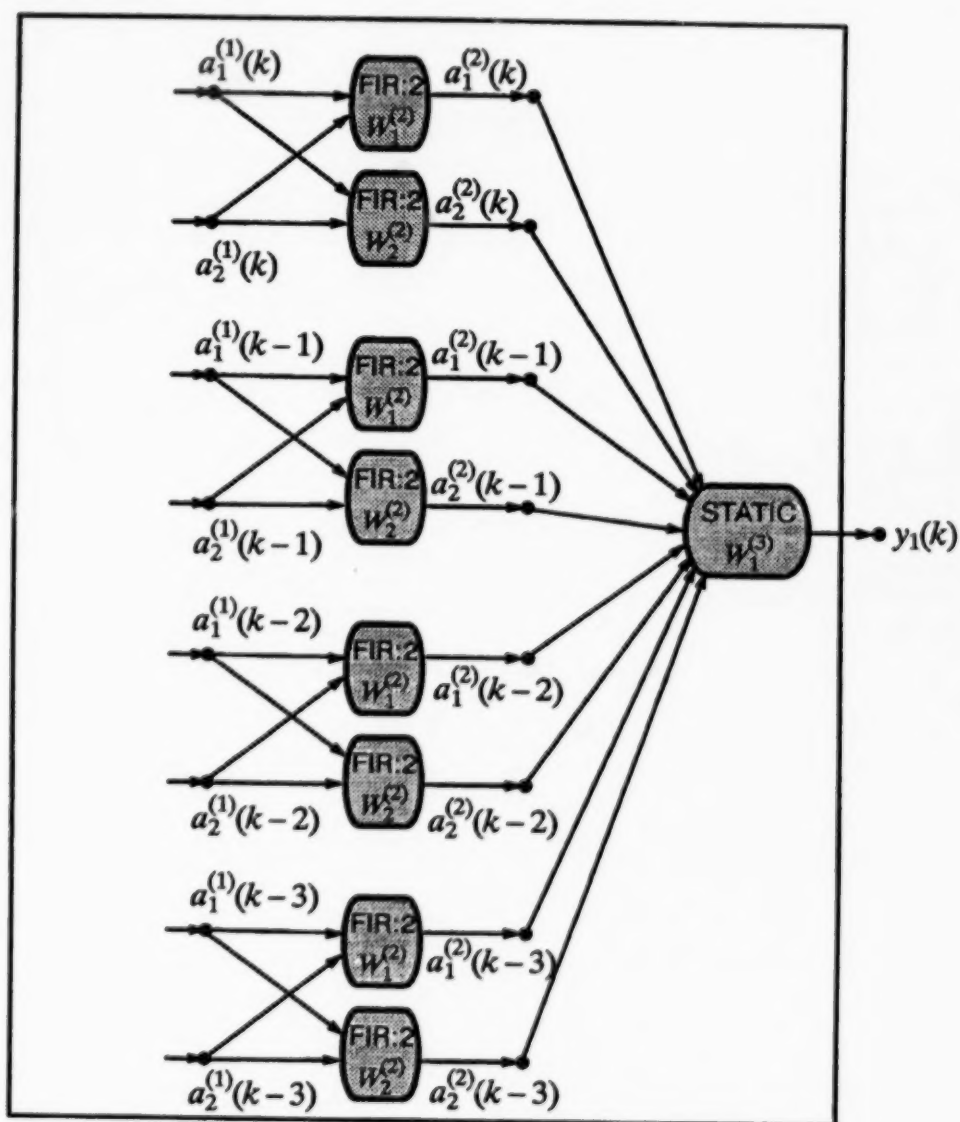


Figure 9: Illustration of the replication of the penultimate layer of processing units required for replacement of the FIR node in the last layer by an equivalent static node.

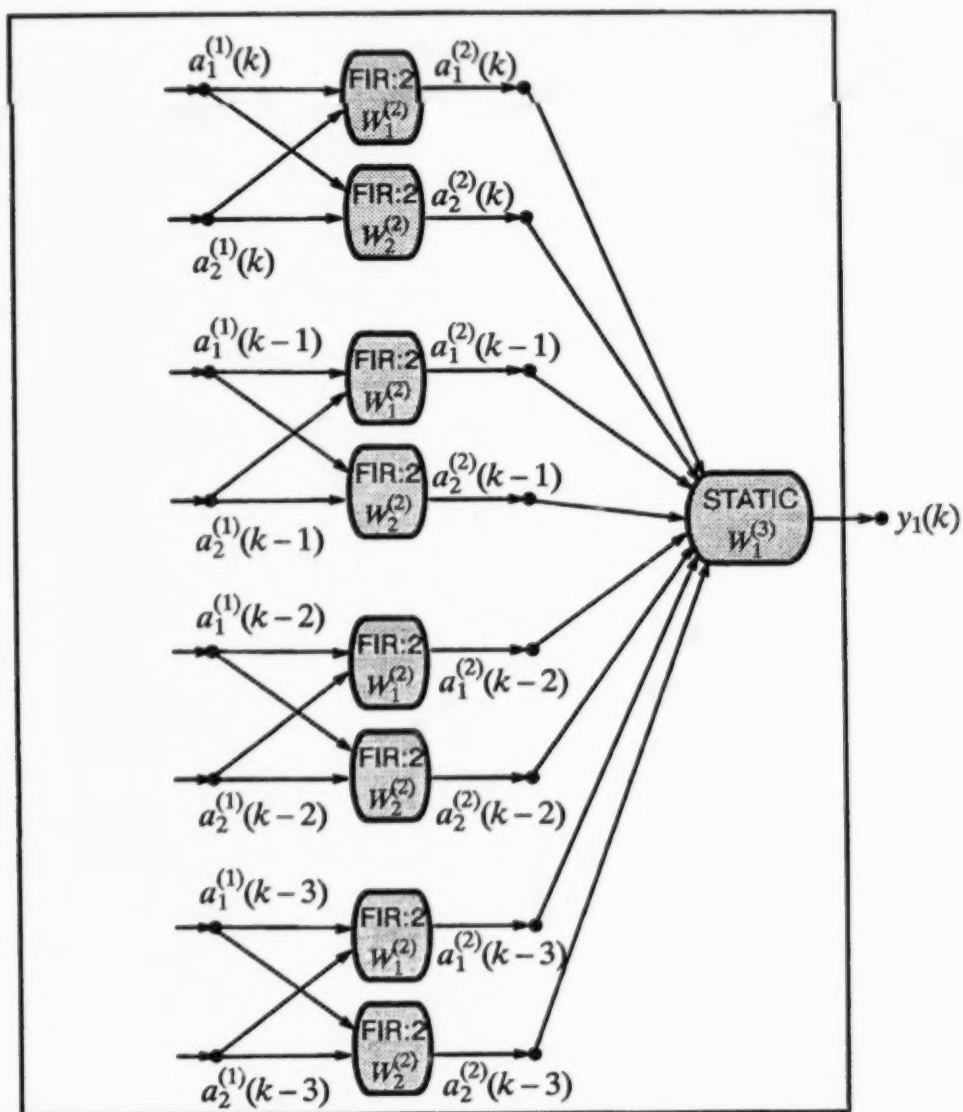


Figure 9: Illustration of the replication of the penultimate layer of processing units required for replacement of the FIR node in the last layer by an equivalent static node.

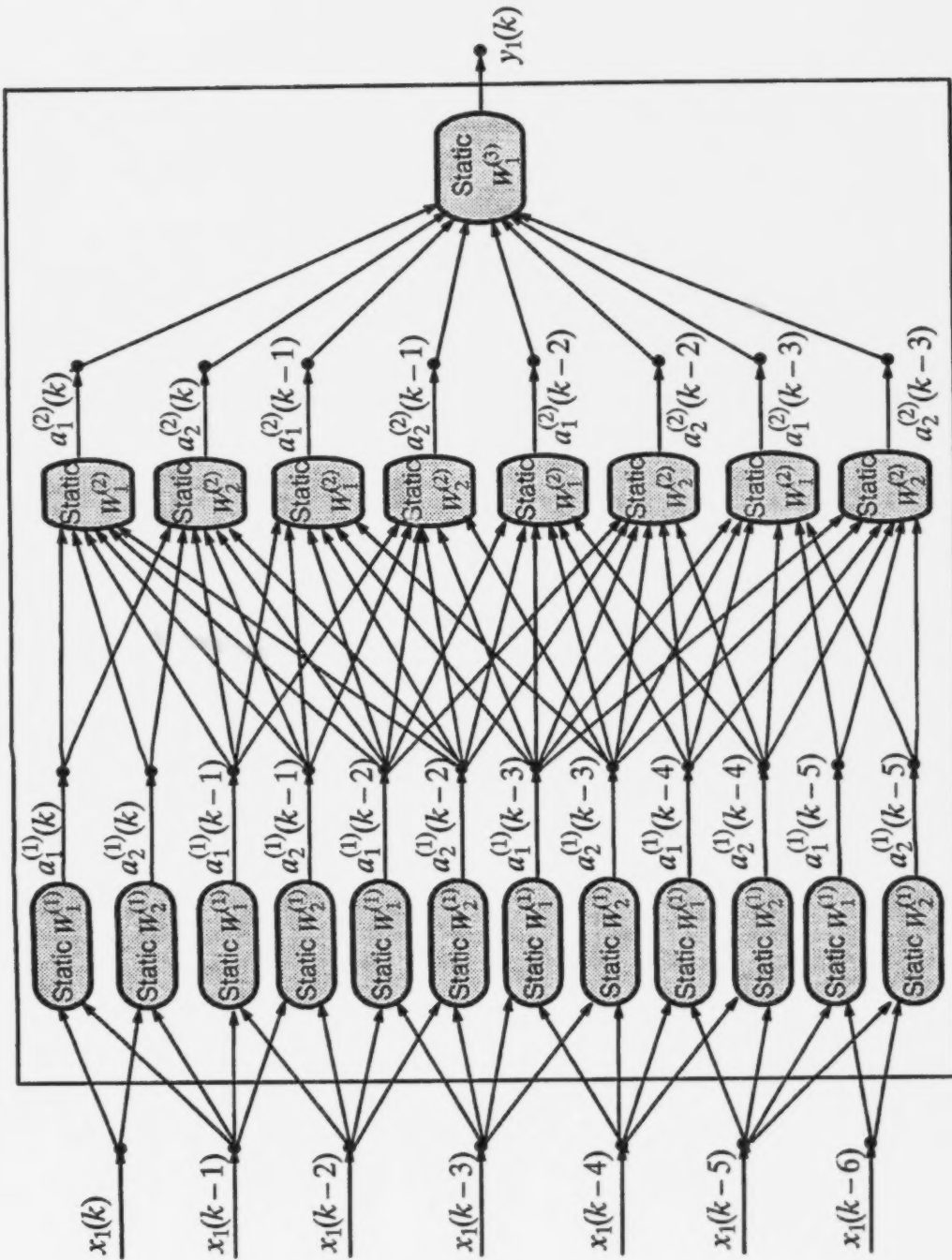


Figure 10: Illustration of the static-network equivalent of the FIR network shown in Figure 5.

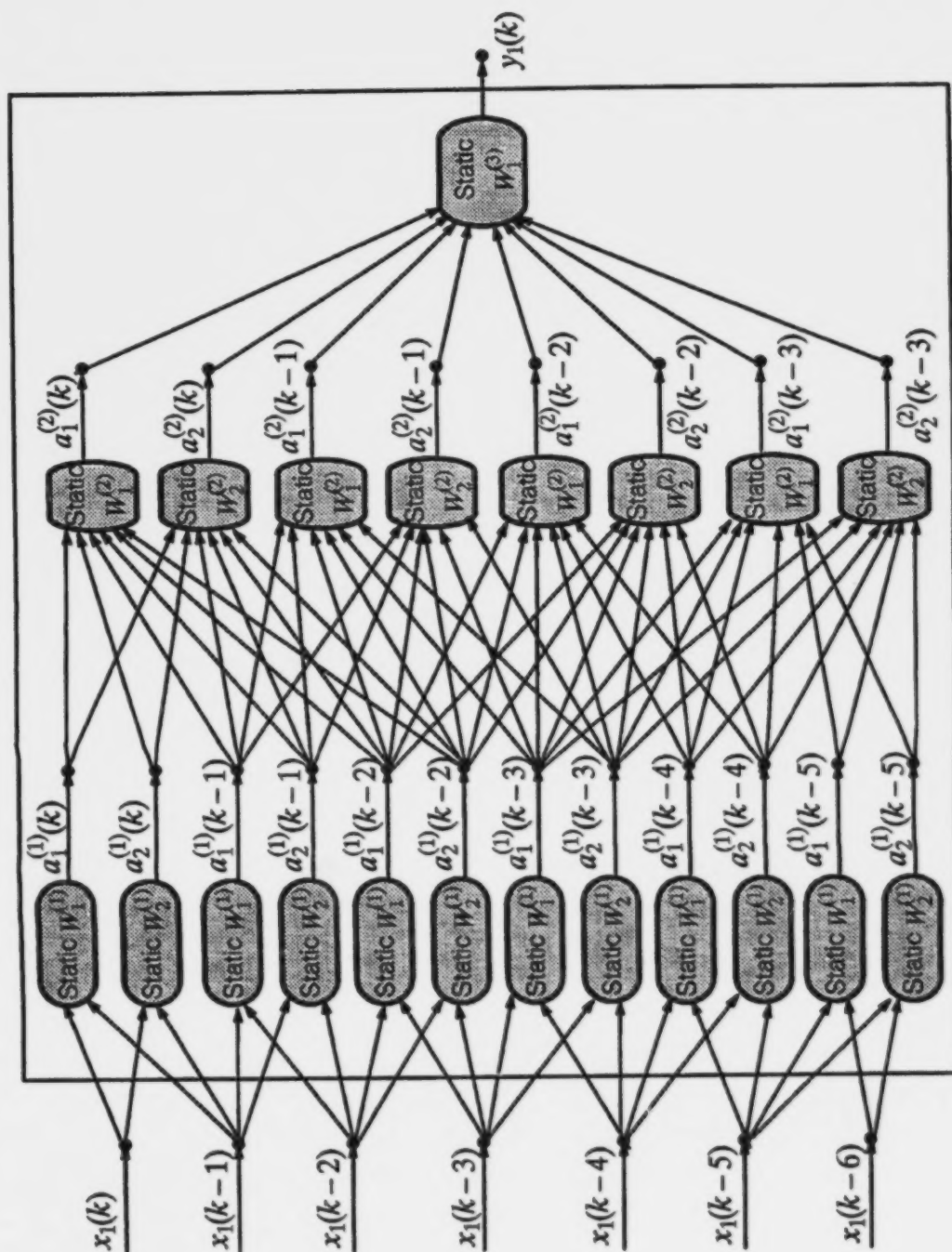


Figure 10: Illustration of the static-network equivalent of the FIR network shown in Figure 5.

3.4 How the FIRNN/TDNN processes temporal information

As discussed above, each processing unit of an FIRNN/TDNN observes a time window of data and thus has the ability to process temporal information. During training, the individual units of the network discover useful temporal events. Loosely speaking, each unit learns to fire whenever it detects the temporal event to which it becomes tuned. Thus each unit operates as a time-invariant feature detector (*i.e.*, each unit learns to detect features independent of the alignment of these features within the input token).

From the unfolded network in Figure 10, we see that the units in the first layer make decisions based on a relatively short time window of input data, whereas units in successive layers make decisions based on successively longer time windows of the input data. In particular, at each instant of time, the outputs of the units in layer ℓ depend on the current and past $M^{(\ell)}$ values of the outputs from the previous layer. Letting $t^{(\ell)}$ denote the duration of the observation window associated with a unit in layer ℓ (*i.e.*, the time span, in iterations, of the input data which influence the instantaneous output of a unit in layer ℓ), we note that $t^{(\ell)} = t^{(\ell-1)} + M^{(\ell)}$ with $t^{(1)} = 1$. Hence, we have that the duration of the observation window associated with the units of layer ℓ is given by:

$$t^{(\ell)} = 1 + M^{(1)} + \dots + M^{(\ell)}, \quad 1 \leq \ell \leq L. \quad (1)$$

Thus, it is the job of the first layer processing units to discover the temporal events of short-duration and the job of the processing units in the successive layers to discover the events of longer duration (possibly concatenations of short-duration events).

3.5 The temporal backpropagation training algorithm

Most neural networks are trained using a gradient-descent type algorithm to minimize a selected cost function. The most commonly used cost function is the sum of squared errors (between desired outputs and actual network outputs); where the sum is taken over all output units and over all patterns in the training set. Standard backpropagation is a very efficient way of implementing gradient descent for static layered feedforward neural networks. The temporal backpropagation training algorithm, for FIR neural networks, is an extension of the standard backpropagation training algorithm for static feedforward neural networks.

In the case of a dynamic neural network, each input pattern (also referred to as an input sequence or token) is a multidimensional sequence. We consider a training set with P pairs of input sequences and their associated desired-response sequences. We assume the input and desired-response sequences to be of length K . The dimension of the input sequences should agree with the number of inputs to the network, $N^{(0)}$; whereas, the dimension of the desired-response sequences should agree with the number of network outputs, $N^{(L)}$. The dummy variable p is used as a pair index; the dummy variable k is used as a temporal index. The vectors $\mathbf{x}^{(p)}(k)$ and $\mathbf{d}^{(p)}(k)$ denote respectively the k th vector of the p th pair of input and desired-response sequences. The individual components of these vectors are referenced using subscripts as shown below:

$$\mathbf{x}^{(p)}(k) = \begin{bmatrix} x_1^{(p)}(k) \\ x_2^{(p)}(k) \\ \vdots \\ x_{n_0}^{(p)}(k) \end{bmatrix} \quad \text{and} \quad \mathbf{d}^{(p)}(k) = \begin{bmatrix} d_1^{(p)}(k) \\ d_2^{(p)}(k) \\ \vdots \\ d_{n_L}^{(p)}(k) \end{bmatrix}.$$

The vector of network outputs immediately following the presentation of the k th input vector of the p th input sequence is denoted by $\mathbf{y}^{(p)}(k)$. It should be noted that although not explicitly indicated, $\mathbf{y}^{(p)}(k)$ is a function of the weights, $\mathbf{w}$, of the network.

We let $\xi(\mathbf{w})$ denote the cost function to be minimized and consider the following definition for $\xi(\mathbf{w})$:

$$\xi(\mathbf{w}) = \sum_{p=1}^P \xi^{(p)}(\mathbf{w}) = \sum_{p=1}^P \left[\sum_{k=1}^K \sum_{j=1}^{N^{(L)}} \frac{1}{2} \left(\underbrace{d_j^{(p)}(k) - y_j^{(p)}(k)}_{e_j^{(p)}(k)} \right)^2 \right]. \quad (2)$$

$\xi^{(p)}(\mathbf{w})$

This cost function is simply the sum-of-squared-errors (scaled, for convenience, by a factor of 1/2) with the sum extending over all sequences in the training set, over the entire duration of each sequence, and over all output units of the network. As with any gradient-descent algorithm, temporal backpropagation updates the weights of the network in the direction of the negative gradient of the cost function with respect to $\mathbf{w}$ evaluated at the

current value of $\mathbf{w}$. Letting $\mathbf{w}(i)$ denote the vector of network weights after the i th weight update, we can express the weight updates mathematically as:

$$\mathbf{w}(i+1) = \mathbf{w}(i) - \mu \nabla \xi|_{\mathbf{w}(i)} \quad (3)$$

where the learning factor, μ , is a positive constant and the notation $\nabla \xi|_{\mathbf{w}(i)}$ can be read "the gradient of ξ with respect to $\mathbf{w}$ evaluated at $\mathbf{w}(i)$."

3.5.1 Derivation of temporal backpropagation

The weight update equation for an individual weight of the network, say one of the weights associated with the FIR filter connecting node m of layer $\ell-1$ to node q of layer ℓ is given by the appropriate component of equation 3, i.e.:

$$w_{qmn}^{(\ell)}(i+1) = w_{qmn}^{(\ell)}(i) - \mu \frac{\partial \xi(\mathbf{w})}{\partial w_{qmn}^{(\ell)}} \quad (4)$$

From equation 2, we see that the partial of $\xi(\mathbf{w})$ with respect to $w_{qmn}^{(\ell)}$ can be expressed as:

$$\frac{\partial \xi(\mathbf{w})}{\partial w_{qmn}^{(\ell)}} = \sum_{p=1}^P \frac{\partial \xi^{(p)}(\mathbf{w})}{\partial w_{qmn}^{(\ell)}} \quad (5)$$

Repeated application of the chain rule yields an efficient procedure for evaluating the partials of $\xi^{(p)}(\mathbf{w})$ with respect to $w_{qmn}^{(\ell)}$. The derivation of this procedure consists of several stages. In the first stage, we apply the chain rule to express the partial of $\xi^{(p)}(\mathbf{w})$ with respect to $w_{qmn}^{(\ell)}$ in terms of the partials of $\xi^{(p)}(\mathbf{w})$ with respect to the signals directly affected⁵ by $w_{qmn}^{(\ell)}$; for convenience, we will refer to these signals as the *signals of stage 1*. In the second stage, we apply the chain rule to express the partials of $\xi^{(p)}(\mathbf{w})$ with respect to the signals of stage 1 in terms of the partials of $\xi^{(p)}(\mathbf{w})$ with respect to the *signals of stage 2* (those signals directly affected by the signals of stage 1). This process continues until reaching a stage for which the signals of the current stage directly affect $\xi^{(p)}(\mathbf{w})$.

5. A given signal is said to be directly affected by a network parameter/signal if that parameter/signal appears explicitly in the equation defining the given signal.

$$\left. \begin{aligned}
s_q^{(\ell)}(k) &= b_q^{(\ell)} + \sum_{m=1}^{N^{(\ell-1)}} \sum_{n=0}^{M^{(\ell)}} a_m^{(\ell-1)}(k-n) w_{qmn}^{(\ell)} \\
a_q^{(\ell)}(k) &= f(s_q^{(\ell)}(k))
\end{aligned} \right\} 1 \leq \ell \leq L$$

$$\xi^{(p)}(\mathbf{w}) = \sum_{k=1}^K \sum_{j=1}^{N^{(L)}} \frac{1}{2} \left(d_j^{(p)}(k) - \underbrace{a_j^{(L)}(k,p)}_{y_j^{(p)}(k)} \right)^2$$

Figure 11: Set of equations describing the dynamics of the FIR neural network.

In the following paragraphs, we present a derivation of temporal backpropagation based on the set of equations in Figure 11 describing the dynamics of the FIR neural network.

The first stage of the derivation begins by noting that the weight $w_{qmn}^{(\ell)}$ directly affects the linear output of the node to which it is attached at every instant of time. Application of the chain rule to this observation yields:

$$\frac{\partial \xi^{(p)}(\mathbf{w})}{\partial w_{qmn}^{(\ell)}} = \sum_{k=1}^K \frac{\partial \xi^{(p)}(\mathbf{w})}{\partial s_q^{(\ell)}(k)} \frac{\partial s_q^{(\ell)}(k)}{\partial w_{qmn}^{(\ell)}} = \sum_{k=1}^K \frac{\partial \xi^{(p)}(\mathbf{w})}{\partial s_q^{(\ell)}(k)} a_m^{(\ell-1)}(k-n) \quad (6)$$

where it is understood that $s_q^{(\ell)}(k)$ is the linear output of node q on layer ℓ just after the k th input vector of the p th input sequence has been presented to the FIR neural network.

To emphasize the fact that a given internal signal of the network (e.g., $s_q^{(\ell)}(k)$) results from the presentation of the p th input sequence of the training set, we will occasionally use the sequence index, p , as a second argument (e.g., we may write $s_q^{(\ell)}(k,p)$).

The goal of the second stage is to express $\partial \xi^{(p)}(\mathbf{w}) / \partial s_q^{(\ell)}(k)$ in terms of the partials of $\xi^{(p)}(\mathbf{w})$ with respect to those signals directly affected by $s_q^{(\ell)}(k)$. Noting that there is only one signal, $a_q^{(\ell)}(k)$, which is directly affected by $s_q^{(\ell)}(k)$, we find:

$$\frac{\partial \xi^{(p)}(\mathbf{w})}{\partial s_q^{(\ell)}(k)} = \frac{\partial \xi^{(p)}(\mathbf{w})}{\partial a_q^{(\ell)}(k)} \frac{\partial a_q^{(\ell)}(k)}{\partial s_q^{(\ell)}(k)} = \frac{\partial \xi^{(p)}(\mathbf{w})}{\partial a_q^{(\ell)}(k)} f'(s_q^{(\ell)}(k)). \quad (7)$$

For the third stage, we consider separately the case when ℓ is the output layer ($\ell = L$) from the case when ℓ corresponds to a hidden layer ($1 \leq \ell < L$). In the case $\ell = L$, we note that $a_q^{(L)}(k)$ directly affects $\xi^{(p)}(\mathbf{w})$, and thus for the sum squared-error cost function of (2), we have that:

$$\frac{\partial \xi^{(p)}(\mathbf{w})}{\partial a_q^{(L)}(k)} = - \left(d_q^{(p)} - a_q^{(L)}(k) \right) = -e_q^{(p)}(k). \quad (8)$$

For the case $\ell < L$, we note that $a_q^{(\ell)}(k)$ directly affects the signals $s_u^{(\ell+1)}(k + \tau)$ corresponding to $0 \leq \tau \leq M^{(\ell+1)}$ and $1 \leq u \leq N^{(\ell+1)}$. Thus:

$$\frac{\partial \xi^{(p)}(\mathbf{w})}{\partial a_q^{(\ell)}(k)} = \sum_{u=1}^{N^{(\ell+1)}} \sum_{\tau=0}^{M^{(\ell+1)}} \frac{\partial \xi^{(p)}(\mathbf{w})}{\partial s_u^{(\ell+1)}(k + \tau)} \underbrace{\frac{\partial s_u^{(\ell+1)}(k + \tau)}{\partial a_q^{(\ell)}(k)}}_{w_{uq\tau}^{(\ell+1)}} \quad (9)$$

As a parallel to standard backpropagation, we introduce the following shorthand notation for $\partial \xi^{(p)}(\mathbf{w}) / \partial s_q^{(\ell)}(k)$:

$$\delta_q^{(\ell)}(k, p) = \frac{\partial \xi^{(p)}(\mathbf{w})}{\partial s_q^{(\ell)}(k)}.$$

We then note that substitution of (9) into (7) yields an efficient procedure for finding the current "delta" of a given node in layer ℓ in terms of the current and future deltas of the nodes in layer $\ell + 1$.

$$\delta_q^{(\ell)}(k, p) = f'(s_q^{(\ell)}(k)) \sum_{u=1}^{N^{(\ell+1)}} \sum_{\tau=0}^{M^{(\ell+1)}} w_{uq\tau}^{(\ell+1)} \delta_u^{(\ell+1)}(k + \tau, p), \quad 0 \leq \ell < L. \quad (10)$$

Substitution of (8) into (7) yields the following initial condition for the recursion of (10):

$$\delta_q^{(L)}(k, p) = f' \left(s_q^{(L)}(k) \right) \left(-e_q^{(L)}(k) \right). \quad (11)$$

From (4), (5), and (6), we can now restate the weight updates for the temporal backpropagation algorithm as follows:

$$w_{qmn}^{(\ell)}(i+1) = w_{qmn}^{(\ell)}(i) - \mu \sum_{p=1}^P \sum_{k=1}^K \delta_q^{(\ell)}(k, p) a_m^{(\ell-1)}(k-n, p) \quad (12)$$

where $\delta_q^{(\ell)}(k, p)$ is calculated according to (10) and (11).

3.5.2 Implementation of temporal backpropagation

The temporal backpropagation algorithm can be summarized as follows:

$$\Delta w_{qmn}^{(\ell)}(i+1) = -\mu \sum_{p=1}^P \sum_{k=1}^K \delta_q^{(\ell)}(k, p) a_m^{(\ell-1)}(k-n, p) \quad (13)$$

where:

$$\delta_q^{(\ell)}(k, p) = \begin{cases} f' \left(s_q^{(\ell)}(k) \right) \left(-e_q^{(\ell)}(k) \right), & \ell = L \\ f' \left(s_q^{(\ell)}(k) \right) \sum_{u=1}^{N^{(\ell+1)}} \sum_{\tau=0}^{M^{(\ell+1)}} w_{uq\tau}^{(\ell+1)} \delta_u^{(\ell+1)}(k+\tau, p), & 0 \leq \ell < L \end{cases} \quad (14)$$

Although not obvious from (13), implementation of the algorithm requires the storage of a limited number of past values of certain network signals (more than are naturally stored by the network in its feedforward mode of operation). This is due to the fact that the delta values of layer ℓ depend on future delta values of layer $\ell+1$. Thus, $\delta_q^{(\ell)}(k, p)$ can not be computed until some time in the future; from (13), we see that calculation of the ℓ th layer weight updates requires storage of $a_m^{(\ell-1)}(k-M^{(\ell)})$ until the time, $k + \varrho^{(\ell)}$, when $\delta_q^{(\ell)}(k, p)$ can be computed.

To gain a more precise understanding of the additional storage required for calculating the weight updates, we next use equation (14) to calculate the delay $\varrho^{(\ell)}$ between time k and the time at which $\delta_q^{(\ell)}(k, p)$ can first be calculated. We see that for $\ell = L$, $\delta_q^{(L)}(k, p)$

can be calculated immediately at time k ; thus $\varrho^{(L)} = 0$. For $\ell = L - 1$, we note from (14) that $\delta_q^{(L-1)}(k, p)$ depends on $\delta_q^{(L)}(k + M^{(L)}, p)$. Since the earliest that $\delta_q^{(L)}(k + M^{(L)}, p)$ can be computed is at iteration $k + M^{(L)}$, the earliest that $\delta_q^{(L-1)}(k, p)$ can be computed is also at iteration $k + M^{(L)}$; thus, $\varrho^{(L-1)} = M^{(L)}$. In general, for $\ell < L$, we note from (14) that $\delta_q^{(\ell)}(k, p)$ depends on $\delta_q^{(\ell+1)}(k + M^{(\ell+1)}, p)$ which can not be computed until iteration $k + M^{(\ell+1)} + \varrho^{(\ell+1)}$; hence $\varrho^{(\ell)} = M^{(\ell+1)} + \varrho^{(\ell+1)}$. Thus, the delay in iterations, $\varrho^{(\ell)}$, between time k and the time at which $\delta_q^{(\ell)}(k, p)$ can first be calculated is given by:

$$\varrho^{(\ell)} = \begin{cases} 0, & \ell = L \\ M^{(\ell+1)} + \dots + M^{(L)}, & 1 \leq \ell < L \end{cases} \quad (15)$$

It is also easy to arrive at equation (15) following a more intuitive approach. We first recall that $\delta_q^{(\ell)}(k, p)$ is a measure of the influence of $s_q^{(\ell)}(k, p)$ on $\xi^{(p)}(\mathbf{w})$. Next, note that there are exactly $\varrho^{(\ell)}$ delay elements between the signal $s_q^{(\ell)}(k, p)$ and the network outputs. This means that $s_q^{(\ell)}(k, p)$ will indirectly influence the network outputs at iterations $k, \dots, k + \varrho^{(\ell)}$, thus, making it impossible to calculate the total influence of $s_q^{(\ell)}(k, p)$ on $\xi^{(p)}(\mathbf{w})$ until iteration $k + \varrho^{(\ell)}$.

Referring back to equation (13), we see that the calculation of the weight updates for the ℓ th layer weights, requires the storage of $M^{(\ell)} + \varrho^{(\ell)}$ past values of the outputs of the units in the $\ell - 1$ st layer; $M^{(\ell)}$ of these past values are naturally stored in the tapped-delay lines of the network; thus, training requires the storage of an additional⁶ $\varrho^{(\ell)}$ past values of the $\ell - 1$ st layer outputs.

6. additional to the past values which are naturally stored by the network.

3.6 Training the FIRNN as a sequence classifier: architecture, desired-response sequences, and error-scaling techniques

When training an FIRNN as a sequence classifier, an appropriate FIRNN architecture is partially defined by the size of the sequences to be classified and the number of classes into which they are to be classified. However, it is not always obvious what to use as a desired-response sequence and/or what type of error-scaling technique to use. Below, we discuss several options.

Consider the case in which we would like to classify sequences of length T and dimension n as belonging to one of m classes. Typically, we choose the total duration of the FIRNN observation window to coincide with the sequence length; this corresponds to choosing an FIR architecture for which the sum, over all layers, of filter orders is $T - 1$. The number of network inputs, $N^{(0)}$, is chosen to correspond to the dimension of the input sequences, n ; and the number of network outputs $N^{(L)}$ is chosen to correspond to the number of possible pattern classes (one output node is associated with each possible pattern class). A suggested architecture for a time-invariant classifier is discussed in section 4.1.1.

To gain insight into an appropriate desired-response sequence, it is useful to determine the desired-response sequence which would coincide with training the equivalent static network as a pattern classifier. The equivalent static network has $N^{(0)} = nT$ inputs (i.e., we now present all T frames of the pattern simultaneously) and $N^{(L)} = m$ outputs. For the purposes of training a static network as a classifier, we would assign desired-response vectors to each pattern in the training set as follows. Each pattern in class c , is assigned a desired-response vector with the c th element equal to 1 and all other elements equal to -1. Standard backpropagation is then applied to prescribe updates to each weight of the network. In order to maintain the equivalence between the unfolded static network and the original FIR network (see section 3.3), certain weights in the unfolded network are constrained to have the same values. To accommodate this constraint, each weight in a group of weights that are constrained to have the same value is updated by an amount equal to the average of the individually prescribed updates.

The reader may check that the weight updates accumulated in the fashion described in the previous paragraph (this is the method used by Waibel *et al.* [45]) are identical with the weight updates which are prescribed by temporal backpropagation if the errors between desired responses and network outputs are evaluated only at the T th iteration (*i.e.*, just as the final frame of an input sequence is presented to the network). The errors for the other iterations may be set to 0 without evaluation (this corresponds to what we refer to as *impulsive error scaling*). We remind the reader that $T - 1$ (the sum of filter orders) iterations are required to completely backpropagate an error vector through the network. Thus, the weight updates resulting from the error vector at iteration T will not be fully accumulated until iteration $2T - 1$. During the final $T - 1$ iterations, zeroes may be clocked into the network while the weight updates are accumulated.

In the previous paragraph, the notion of error-scaling was introduced. Error-scaling is a technique which allows the errors (the differences between desired responses and actual outputs) occurring at certain instants of time to be weighted more heavily than others. Note that scaling the errors at iteration k by $\text{error\_scale}[k]$ (prior to backpropagation of the errors through the network) results in an algorithm which attempts to minimize the following cost function instead of (2):

$$\xi(\mathbf{w}) = \sum_{p=1}^P \xi^{(p)}(\mathbf{w}) = \sum_{p=1}^P \left[\sum_{k=1}^K \sum_{j=1}^{N^{(2)}} \frac{1}{2} \text{error\_scale}[k] \left(d_j^{(p)}(k) - y_j^{(p)}(k) \right)^2 \right]. \quad (16)$$

The impulsive error-scaling technique discussed earlier corresponds to using the following error-scaling sequence:

$$\text{error\_scale}[k] = \begin{cases} 0, & k \neq T \\ 1, & k = T \end{cases} \quad (17)$$

Other error-scaling sequences that we have experimented with include a flat error-scaling sequence that places equal emphasis on errors at all instants of time and a gaussian-shaped error-scaling sequence with peak at iteration T . Experimental results are discussed in [1].

In the case of the impulsive error-scaling sequence, only the desired responses for iteration T influence the resulting weight updates. For the other two types of error-scaling sequences suggested above, one must choose an appropriate desired-response sequence

with length corresponding to the number of iterations ($2T - 1$) required to clock an input sequence in and out of the network. For instance, it seems reasonable to expect that as the input sequence is clocked into the network, the network output associated with the pattern's true class will gradually increase, reaching a peak value of one when the pattern completely fills the network; as the pattern gets clocked out of the network, we would expect to see the network output associated with the pattern's true class begin to fall back to a value of -1. Thus, we may decide to train the network using a desired-response sequence for the true-output node that rises from -1 to 1 as the input pattern is clocked into the network and decreases back to -1 as the pattern is clocked out of the network.

A fundamentally different approach may be to argue that we would like the network to recognize the sequence as soon as possible; however, most emphasis should be placed on recognizing the sequence for iterations close to T . To implement these ideas, we may choose to use flat desired-response sequences (a sequence of 1s for the true-output node and a sequence of -1s for all other output nodes) in conjunction with a gaussian-shaped error scaling function which peaks at iteration T .

4 Successful Applications of FIR/TD NN's

There are two main problem areas to which FIR/TD neural networks have been successfully applied: pattern recognition (especially problems in speech and phoneme recognition) and nonlinear time series prediction.

4.1 Speech and phoneme recognition

Although great advances in automatic speech recognition have been achieved during the past few decades, automated systems do not come close to achieving human-like performance. Speech recognition poses many challenging problems; most of these problems stem from the large amount of variability that exists between signals representing the same event (whether this event be at the phonetic level, the word level, *etc.*). The variability is due to differences between speakers, differences in speed of pronunciation, differences in neighboring acoustical events, *etc.* Most researchers are now focussing on simplified speech-recognition problems; simplification is accomplished by eliminating some of the variability (*e.g.*, by restricting the scope to a single-speaker, to recognition of words uttered in isolation, or to recognition of low-level acoustical events). In the last decade, neural networks have emerged as a promising tool for automatic speech recognition. An excellent review can be found in [30].

A problem that has attracted the interests of the neural network community is that of phoneme classification. A phoneme classifier is an example of a system whose performance is greatly enhanced by the ability to discover and exploit the underlying temporal structure of each signal class. As is the case for speech recognition, the phoneme classification problem is challenging due to the large amount of variability between signals (or tokens) representing the same phonetic class. To cope with this, a successful classifier must be capable of implementing complex nonlinear decision boundaries.

Another challenge which is posed by phoneme classification is the necessity of precise time alignment (*i.e.*, proper segmentation of the signal from which the phoneme was extracted). Segmentation of the signal is a task which is normally assigned to a preprocessor. Very often, the segmentation is done manually (by human intervention). Proper segmentation requires identification of the beginning and end of a phoneme. Precise identification of the beginning and end points of a phoneme is a challenging task in itself due to the large variability introduced by the different phonetic contexts out of which the tokens can be

extracted. Variations in phonetic context are caused by different articulatory movements (caused by different speaking rates) and to coarticulation, which causes the features of one phoneme to be spread over neighboring phonemes [8]. Unfortunately, improper segmentation of the signal often results in poor classification results.

4.1.1 Suggested TDNN/FIRNN architecture for time-invariant speech/phoneme classification

The preceding two paragraphs point to the need for a classifier which is capable of discovering and exploiting the underlying temporal structure of the various phoneme classes, capable of implementing complex decision boundaries, and capable of translation-invariant event detection. The FIR/TDNN possesses these properties. The capability of translation-invariant event detection is discussed somewhat in section 3.4; the ability of the FIR/TDNN to cope with poor segmentation of the signal is elaborated on in the following paragraph.

The architecture of the TDNN allows for successful classification of phonemes despite a variety of alignments of the phoneme within the training and testing tokens (*i.e.*, despite poor segmentation). As discussed in section 3.4, each unit of the network learns to detect useful temporal events independent of the alignment of the event within the token; first-layer units detect short-duration events, units in successive layers detect events of longer duration. Thus, it is natural to expect that the TDNN should be capable of recognizing phonemes independent of their alignment within a token. The challenge is to accommodate the fine-tuning of the hidden-layer units while training the output units to be relatively sloppy feature detectors; note that if the training tokens include a variety of alignments (due to poor segmentation), then we are training the appropriate output of the network to be high for a variety of alignments, and thus cannot expect the output of the network to be a precise indicator of the true alignment of the phoneme.

Waibel *et al.* [45] suggested that the solution to the challenge mentioned in the previous paragraph is to use a TDNN architecture for which the number of units in both the output and penultimate layers is given by the number of phonetic classes. Rather than being fully connected to the units in the penultimate layer, each output unit is connected only to the corresponding unit in the penultimate layer. The weights of the FIR filter connecting a unit in the output layer to its associated penultimate-layer unit are nonadaptable and all

set equal to the same constant; this means that the final-layer outputs are moving averages of the penultimate-layer outputs. The network is trained so that the appropriate⁷ output unit is high and all other output units are low at the instant of time corresponding to the presentation of the final frame of a token to the network.

The philosophy behind this suggested architecture is that the output of the appropriate penultimate-layer unit will learn to go high at a time corresponding to the true alignment of the phoneme. Since the corresponding output-layer unit is a moving average of this penultimate-layer unit, its output will be a smeared version of the penultimate-layer output (thus, being less indicative of the phoneme alignment yet allowing the phoneme to be correctly classified in spite of time misalignments).

4.1.2 Application of the TDNN/FIRNN to phoneme recognition

Recently, Waibel *et al.* [45,46] have successfully applied a TDNN to the task of speaker-dependent phoneme recognition. For this application, they had three male professional announcers utter, in isolation, words from a large Japanese database. Training and testing tokens containing the phonemes "B", "D", and "G" were then extracted from the utterances; this resulted in three training and testing sets (one pair for each speaker). Each training and testing set consisted of between 600 and 700 tokens.

Each preprocessed token consisted of 16 normalized melscale spectral coefficients for each of 15 time frames. Each time frame represented 10 ms of speech and each token represented a time window of 150 ms duration.

For classification, they used a three-layer TDNN with eight units on the first layer, three units in the second or penultimate layer, and three units in the third or output layer. The first-layer units were fully connected to the 16 inputs (spectral coefficients) with second-order FIR filters; the second-layer units were fully connected to the first-layer outputs with fourth-order FIR filters; and each output unit was connected to the corresponding penultimate-layer output with an eighth-order FIR filter. Using the notation of section 3.2, the architecture of this neural network can be concisely described as: $\text{FIR}[(16:0) \times (8:2) \times (3:4) \times (3:8)]$. It should be noted that their choice of filter order for

7. When training a network as a classifier, it is typical to choose the number of outputs equal to the number of classes. Thus, each output unit is associated with one class. The network is trained so that when a pattern from a given class is presented, the associated output unit will go high.

the first-layer units was motivated by an earlier study that suggested 30 ms as a typical duration⁸ for low level acoustic-phonetic events which were useful for their phoneme recognition problem. The observation window of the second-layer units was 70 ms (suggesting that this is a sufficient observation period for correct classification of a token containing a properly aligned phoneme). Likewise, the observation window of the third-layer units was 150 ms (the entire token). The additional 80 ms observed by the output units allows the classifier to tolerate time misalignments. Although performance deteriorated with increased misalignments, Waibel *et al.* found they could tolerate misalignments of ± 30 ms (they quoted a 2.6 percent increase in error rate when all training tokens were artificially shifted by 20 ms).

Currently, the best performing speech recognition techniques use Hidden Markov Models (HMM's) [30]. Waibel *et al.* have found that the performance of the TDNN compares favorably with the performance of the HMM for their phoneme recognition task. The recognition rate (averaged over the three test sets) was 98.5% for the TDNN compared to 93.6% for the HMM.

Perhaps the biggest concern which arose out of the initial work of Waibel *et al.* was the amount of time required for training the network and how this would scale with problem size. In a later study [46], these concerns were eased when Waibel found that modular training of a network for a bigger problem was feasible.

4.2 Application of the FIRNN to problems in time series prediction

In addition to problems in pattern recognition, the FIR neural network has been successfully applied to problems requiring nonlinear time-series modelling. These include problems in time-series prediction such as prediction of exchange rates and the prediction of a variety of simulated and real-life chaotic time series [47]. The time-series prediction problem can be stated as follows. Given the first K values of a time series, find a model which accurately predicts the $K + 1st$, $K + 2nd$, ... values of the series.

As discussed by Weigend *et al.* [50], "deterministic chaos is characterized by an exponential divergence of nearby trajectories." Although long-term predictability of a chaotic

8. Note that the observation window of the first-layer units is 30 ms; each unit observes 3 time frames of data and each time frame represents 10 ms.

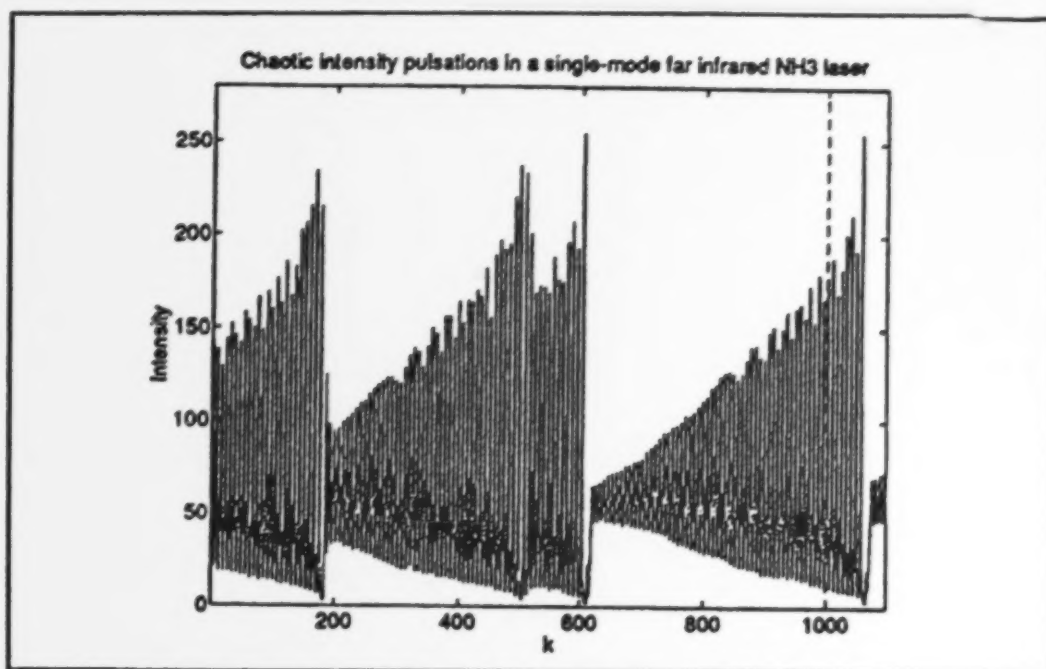


Figure 12: 1100 time points of chaotic laser data from the Santa Fe Institute Time Series Prediction Competition (reproduced with permission from [47]).

time series is impractical, short-term predictability is possible and requires a prediction algorithm capable of capturing the short-term structure of the series.

The FIR neural network gained recognition as a powerful tool for time series prediction when it produced the winning entry of the Santa Fe Institute Time Series Prediction and Analysis Competition⁹ (held in the fall of 1991) [51]. For this competition, the contestants were given the first 1000 points of measured data representing chaotic intensity pulsations of an NH_3 laser and were asked to predict the next 100 points of the series (the series is shown in Figure 12). The source of the time series was withheld from the contestants until after the competition. Entries were based on a variety of methods; in addition to the FIR neural networks, the methods included: k - d trees, piecewise linear interpola-

9. The Time Series Prediction and Analysis Competition was established as a means of evaluating and benchmarking new and existing techniques for time-series prediction. The winning entry to the 1991 competition was submitted by Eric A. Wan [47].

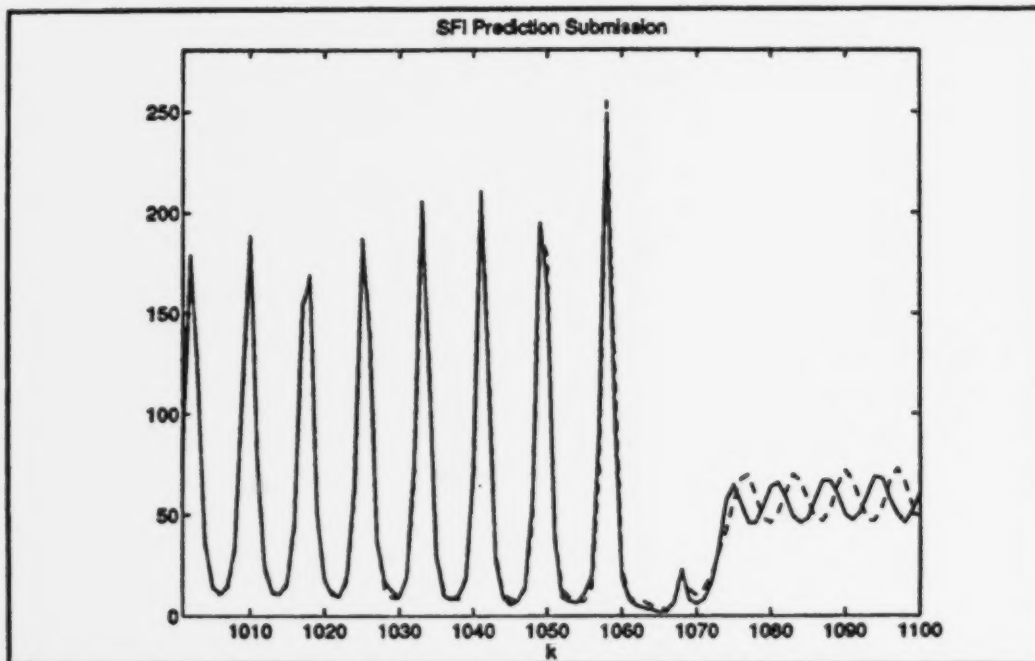


Figure 13: Comparison of the FIRNN Iterated Prediction (solid line), based on the first 1000 points of the laser series, to the actual continuation of the series (dashed line). Reproduced with permission from [47].

tion, low-pass embedding, singular value decomposition, nearest neighbors, Wiener filters, standard multilayer feedforward neural networks and recurrent neural networks.

As noted in [47], the collapses of the intensity make it a difficult prediction problem. The problem is even more challenging due to the fact that the training set contained only three examples of such collapses. The winning entry, was obtained by Wan using a $\text{FIR}[(1:0) \times (12:25) \times (12:5) \times (1:5)]$. The iterated prediction of the FIR neural network is compared to the true extension of the series in Figure 13. In contrast to other methods, the FIR network predicted the next collapse very accurately; the last 25 of the 100 predicted points were somewhat out of phase.

As noted by Wan, the choice of architecture was obtained using some heuristics in addition to a trial-and-error type approach. The heuristics which were used to choose initial values for filter orders were based on the autocorrelation function of the laser data and the prediction errors which were obtained using linear autoregressive filters (AR) of various orders. The autocorrelation function indicated significant correlation up to lags of

40 or so, whereas the prediction errors for the linear AR filters showed significant error reduction due to increased filter order up to order eight followed by a further gradual error reduction up to order 25.

Of the 1000 data points which were supplied to the contestants, Wan used the first 900 for training and the last 100 were used as a cross-validation set to avoid over-training. It should be noted that the chosen network had more variable parameters (1105) than there were data points; although this situation should normally be avoided, techniques such as cross validation help to ensure generalization.

For comparison purposes, Wan used the same data to train (after the competition) a couple of standard multilayer neural networks. He trained these networks to act as one-step predictors using at each step, the most recent 35 past values as inputs (this was based on the total filter order of 35 for Wan's FIR network). The first network he trained was a $(35 \times 12 \times 12 \times 1)$ and thus had the same number of processing units as his FIR network. The other network was a $(35 \times 20 \times 20 \times 1)$ and was chosen due to the fact that it had roughly the same number of weights as the FIR network. Both networks performed much worse than the FIR network and neither was able to predict the next intensity collapse.

5 Review of Dynamic Neural Networks

In addition to the FIR/TDNN architecture, there exists a plethora of other dynamic network architectures which are well-suited for temporal-processing tasks. Dynamic architectures can be categorized as feedforward (with memory) or recurrent. The FIR/TDNN is an example of a dynamic feedforward architecture. In the normal mode of operation, the signal flow of a dynamic feedforward network is always forward through the network; there are no feedback paths. On the other hand, a recurrent neural network has at least one feedback path. As described by Connor *et al.* [6], recurrent neural networks can be broadly classified as *relaxation* or *standard*.

Relaxation networks include Boltzman Machines [21] and Hopfield networks [23, 22]; they are often used for data-association and optimization tasks. The typical mode of operation is to start them running in a given state and to let them run until settling to a stable state¹⁰. Of interest to some readers may be the work of Silven [42] who has applied a Hopfield network to the problem of multiple-target tracking. The main focus of this review will be standard recurrent networks; no additional comments will be made regarding relaxation networks.

Standard recurrent networks are typically used to model trajectories and can be subdivided according to the type of recurrence. For example, a locally recurrent network implies that the feedback paths are contained within individual nodes of the network (e.g., the output of each node of the network might be fed back as an additional input to the node); whereas a partially recurrent network¹¹ is one for which at least one feedback path involves multiple nodes (for example, the outputs of a network may be fed back as additional inputs to the network); and a fully recurrent network¹¹ is one for which the output of every node is used as an input to all nodes in the network.

Descriptions of various dynamic neural network architectures as well as problems to which they have been applied are provided below. Dynamic feedforward networks are described in section 5.1, locally recurrent globally feedforward networks are described in section 5.2, the fully recurrent network is described in section 5.3, and some examples of partially recurrent networks with application to nonlinear autoregressive moving-average

10. It is possible that they will never reach a stable state.

11. Definitions of partially recurrent and fully recurrent network may vary from author to author.

(NARMA) modelling are described in section 5.4. Section 5.5 presents the results of a preliminary comparison of various recurrent networks conducted by Tsoi and Back[44], and section 5.6 briefly discusses the two classes of training algorithms which are most frequently used for training dynamic networks.

5.1 Dynamic feedforward neural networks

Besides the FIR/TDNN, there are several other neural network paradigms which fall under the category of dynamic feedforward neural networks. The Time-Derivative neural network and the Tempo 2 neural network are two examples and are described below.

5.1.1 The time-derivative neural network architecture

Paliwal [35] has introduced the time-derivative neural network as an alternative to the time-delay neural network used by Waibel *et al.* [45]. For this network, the temporal context of the signals is conveyed using time derivatives as opposed to time delays; each node is presented with the outputs of the previous-layer nodes as well as the first D derivatives of these outputs. The dth derivative of a signal at time t is computed as the dth orthogonal polynomial coefficient [9] over a window of length $2L + 1$ centered at time t .

A possible advantage of the time-derivative neural network over the time-delay neural network is that the window of time observed by a given processing unit can be lengthened without an associated increase in the number of weights. This is done by increasing the number of time frames, $2L + 1$, over which the derivatives are evaluated. Note that the number of weights associated with a given processing unit is determined by the number of nodes in the previous layer and the number of derivatives, D , representing the outputs of these nodes; the time duration, $2L + 1$, over which a derivative is evaluated does not affect the number of weights.

Paliwal has applied the time-derivative neural network to the task of speaker-independent isolated-word recognition using a vocabulary consisting of nine English e-set alphabets: B, C, D, E, G, P, T, V, and Z. The training and testing sets consisted respectively of 24 and 40 utterances per word from four speakers (two male and two female). The pre-processing resulted in a vector of 12 lifted cepstral coefficients for each time frame (15 ms) of speech. The performance of the time-derivative neural network was evaluated as a function of the duration of the temporal context (*i.e.*, the time duration over which the

derivatives are evaluated), and the number of higher-order derivatives which were used. In all cases, the time-derivative neural network consisted of 16 hidden units and 9 output units (one for each class). At each instant of time, each hidden unit received as input the 0th through D th-order derivatives of the 12 cepstral coefficients representing the current time frame. The derivatives were evaluated over $2L + 1$ time frames (or over a period of time equal to $(2L + 1) \times 15$ ms).

The experimental results (for $D = 1$) showed that the performance of the time-derivative neural network continued to increase with increasing duration of temporal context up to a duration of 195ms ($L = 6$), which yielded a recognition accuracy of 76.1%. Further increase in L resulted in a decreased recognition accuracy. Experiments were then conducted with $L = 6$ to investigate the performance accuracy as a function of the number of derivatives which were used. The recognition accuracy increased to 78.6% when D was increased to two, but fell back to 77.2% when D was further increased to three.

Paliwal then compared the performance of the time-derivative neural network to that of the time-delay neural network. The architecture of the time-delay neural network was identical (in terms of the number of processing units in each layer) to that of the time-derivative neural network; namely, the twelve cepstral coefficients for the current time-frame were fed to each of the 16 hidden-layer units which in turn fed the 9 output units. All time delays were associated with the hidden-layer units; the output units were static. Performance of the TDNN was evaluated as a function of the number of delays incorporated by the hidden-layer units. Experiments were conducted using TDNNs employing from one to five time delays in the units of the hidden layer. The performance of the TDNN's with three, four, and five delays was very similar (recognition accuracies of 67.3%, 67.5%, and 67.5% respectively); thus the performance of the TDNN with three delays was compared to the performance of the best time-derivative neural network.

Although the recognition accuracy of the TDNN with three delays was 11.3% lower than the accuracy achieved by the best time-derivative neural network (corresponding to $L = 6$ and $D = 2$), there are several points to be made. First, the duration of the temporal contexts of the two networks are quite different; the duration of the temporal context of a TDNN with three time delays is 60 ms (four time-frames worth of data), compared

to 195 ms (13 time-frames of data) for the best time-derivative neural network. Indeed, the results provided by Paliwal reveal that the time-derivative neural network with a temporal context of 75ms (the smallest duration reported for the time-derivative network) achieved a recognition accuracy of only 67.2% (this result corresponds to $L = 2$ and $D = 1$) which is comparable to the performance of the TDNN with a temporal context duration of 60 ms. Since Paliwal reported results for the TDNN only up through a temporal context duration of 90 ms (much smaller than the 195 ms duration used by the best time-derivative neural network), it is unclear whether or not the performance of the TDNN would have reached or even surpassed that of the time-derivative neural network had it been allowed a comparable temporal-context duration. Second, the generalization ability of a network, as measured by the performance on the test set, usually deteriorates as the number of weights becomes large compared to the number of patterns in the training set. In the cases discussed above, the TDNN (with three time delays) had 937 associated connection weights and the best time-derivative neural network ($L = 6$ and $D = 2$) had 745 connection weights. Since increasing the duration of the temporal context for the TDNN to 195 ms would result in a TDNN with 2665 weights, the existing training set would likely be too small to properly train the network. Thus, although it is possible that the TDNN might outperform the time-derivative neural network were it allowed a comparable temporal-context duration, the number of weights required for the TDNN network would be much larger than for the time-derivative neural network.

5.1.2 The tempo 2 network

The Tempo 2 network of Bodenhausen and Waibel [4] allows for the automatic adjustment of the temporal context for each connection between the input and hidden-layer units. More specifically, each input is connected to each of the hidden layer units via a gaussian-shaped kernel or filter; the width (standard deviation) and the location (mean) of the kernel function are adaptable parameters which can be adjusted using a gradient descent procedure. The linear output of each hidden-layer unit is a weighted sum of the outputs from the connecting filters:

$$s_j^{(1)}(k) = \sum_{q=0}^{N^{(0)}} w_{jq}^{(1)} \sum_{n=0}^k x_m(n) G(n, k, m_{jq}, \sigma_{jq})$$

where the gaussian kernel, $G(n, k, m_{jq}, \sigma_{jq})$, connecting the q th input to the j th hidden-layer unit is defined as:

$$G(n, k, m_{jq}, \sigma_{jq}) = \frac{1}{\sqrt{2\pi\sigma_{jq}^2}} \exp\left(-\frac{(n-k+m_{jq})^2}{2\sigma_{jq}^2}\right).$$

During learning, the parameters $w_{jq}^{(1)}$, m_{jq} , and σ_{jq} are adjusted so as to minimize the selected cost function ξ . Using the chain rule, we obtain the following update rules for the various parameter types:

$$\Delta w_{jq}^{(1)} = -\mu_w \frac{\partial \xi}{\partial w_{jq}^{(1)}} = -\mu_w \frac{\partial \xi}{\partial s_j^{(1)}} \frac{\partial s_j^{(1)}}{\partial w_{jq}^{(1)}} = -\mu_w \frac{\partial \xi}{\partial s_j^{(1)}} \sum_{n=0}^k x_q(n) G(n, k, m_{jq}, \sigma_{jq})$$

$$\Delta m_{jq} = -\mu_m \frac{\partial \xi}{\partial m_{jq}} = -\mu_m \frac{\partial \xi}{\partial s_j^{(1)}} \frac{\partial s_j^{(1)}}{\partial m_{jq}} = -\mu_m \frac{\partial \xi}{\partial s_j^{(1)}} w_{jq}^{(1)} \sum_{n=0}^k x_q(n) \frac{\partial G(n, k, m_{jq}, \sigma_{jq})}{\partial m_{jq}}$$

and

$$\Delta \sigma_{jq} = -\mu_\sigma \frac{\partial \xi}{\partial \sigma_{jq}} = -\mu_\sigma \frac{\partial \xi}{\partial s_j^{(1)}} \frac{\partial s_j^{(1)}}{\partial \sigma_{jq}} = -\mu_\sigma \frac{\partial \xi}{\partial s_j^{(1)}} w_{jq}^{(1)} \sum_{n=0}^k x_q(n) \frac{\partial G(n, k, m_{jq}, \sigma_{jq})}{\partial \sigma_{jq}}$$

where the constants μ_w , μ_m , and μ_σ are learning factors for the weights, locations, and widths of the filter connections respectively. Note also that Backpropagation can be used to evaluate the derivative of the cost function with respect to the linear output, $s_j^{(1)}$, of a hidden-layer node.

The temporal processing capabilities of the Tempo 2 network are very powerful. In addition to the fact that the location, width, and associated weight of each connecting filter can be adjusted using gradient descent, Bodenhausen and Waibel have developed a procedure which allows additional connections between units to be established as training proceeds. The network is initialized with only one connecting filter between a given input and a hidden-layer node; during training, the location of the connecting windows are monitored; once a certain level of performance has been reached, those connections with filter locations which continue to move back and forth are split (replaced by two new connections) and then trained independently. Thus, each hidden unit has the ability to process non-consecutive segments of the input sequences. The ability to tune the width of each

connection allows the tradeoff between increased robustness to time shifts (in order to accommodate misaligned tokens) and decreased time resolution to be evaluated in an automated fashion.

Bodenhausen and Waibel have applied the Tempo 2 network to the same phoneme recognition task to which Waibel *et al.* [45] applied the TDNN. They report results for five experiments in which selected sets of parameter-types were adapted; the sets of adaptable parameters for the five experiments were: {weights}, {locations}, {widths}, {locations and widths}, {weights and locations}, and {weights, delays, and widths}. All of their results are based on a Tempo 2 architecture with eight units in a single hidden layer. All connecting filters were initialized to have constant widths and with random weights and locations.

As may have been expected, the ability to adjust the weights of each connecting filter was more important than the ability to adjust the locations or the widths. (It should be kept in mind that the initial widths of the connecting filters were well suited to the problem due to Waibel's previous experience with the TDNN.) The recognition rate on the test set was 89.3%, 63.0%, and 61.8% for adaptable parameter-type sets of {weights}, {locations}, and {widths} respectively. Despite the importance of the weights, it was interesting to see that the network was able to achieve a recognition accuracy as high as 68.6% for the adaptive parameter set which consisted of the locations and widths of the filter connections (*i.e.*, for this experiment the weights were nonadaptable; they were fixed to their initial randomly-assigned values). The network achieved a recognition accuracy of 97.8% for the adaptive parameter-type set which included weights and locations, and increased slightly further to 98% when all three parameter types were made adaptable.

The recognition accuracy of 98% achieved by the Tempo 2 network with eight hidden units was comparable to that achieved by the TDNN with a manually optimized architecture. The average¹² TDNN performance was 98.5%. Unfortunately, no mention was made of typical training times or numbers of adaptive parameters in the trained network. The ability to automate the process of selecting a suitable temporal context for a task is

12. Waibel *et al.* [45] trained three TDNN's for each of three speakers and achieved recognition accuracies of 98.8%, 99.1%, and 97.5% respectively. It is unclear which speaker was used for the Tempo 2 experiments.

important; this can be a very laborious and time-consuming task when done manually.

5.2 Locally recurrent globally feedforward (LRGF) neural networks

A locally recurrent globally feedforward (LRGF) neural network is one for which feedback paths may exist within individual processing units yet the data flow between processing units is feedforward from processing units in one layer to the next. As discussed by Tsoi and Back [44], there are three basic feedback-types which can be incorporated within a single processing unit: local synapse feedback, local activation¹³ feedback, and local output feedback. Local synapse feedback implies that feedback exists within the synapses (the connections between units); this is the case when the synapses take the form of infinite impulse response (IIR) filters. Local activation (output) feedback implies one or more of the unit's inputs have a linear dependence on past values of the unit's linear (nonlinear) output.

5.2.1 Neural networks with IIR synapses (Back-Tsoi)

Tsoi and Back [44, 2] have investigated replacing the static connections of standard feedforward neural networks with adaptive IIR filters, having transfer functions of the form:

$$\frac{B(z)}{1-A(z)} = \frac{\sum_{m=0}^{M_b} b_m z^{-m}}{1 - \sum_{m=0}^{M_a} a_m z^{-m}} \quad (18)$$

This architecture amounts to a LRGF network with synaptic feedback.

In [2], they compare the ability of a neural network with IIR synapses to one with FIR synapses to model the following nonlinear relationship from the sequence $x(t)$ to the sequence $y(t)$:

$$y(t) = \sin(\pi h(q)x(t)) .$$

For the input sequence $x(t)$, they used a zero-mean white noise source which was low-pass

13. Tsoi *et al.* [44] use the term activation to refer to the linear output of a processing unit; depending on the author, the phrase "activation of a processing unit" might also be used to refer to the (nonlinear) output.

filtered with a cut-off frequency of 7 rad/sec. The operator $h(q)$ was defined as:

$$h(q) = \frac{.99q^{-1}}{1 - .8227q^{-1} + .9025q^{-2}}$$

with q^{-1} being the unit delay operator.

For this task, the performance of an FIR[(1 : 0) × (6 : 11) × (1 : 1)] network was compared to that of an IIR network with 6 hidden-layer nodes and 1 output node. The connections to the hidden-layer nodes were IIR filters with $M_b = 5$ and $M_a = 6$ [see (18) for definitions of M_b and M_a]; whereas, the connections to the output layer nodes were IIR filters with $M_b = 0$ and $M_a = 1$. The mean square error for the test set was 0.0664 for the FIR network and 0.0038 for the IIR network; the variance of the error was 0.0082 for the FIR network and 0.000012 for the IIR network.

5.2.2 Frasconi-Gori-Soda local feedback networks

Frasconi, Gori, and Soda [12] have investigated special cases of LRGF networks employing processing units with local activation feedback as well as those with local output feedback. In particular, they consider the special case of a two-layer network for which the linear output of the j th unit in layer ℓ is given by:

$$s_j^{(\ell)}(k) = \begin{cases} w_{jj}^{(\ell)} s_j^{(\ell)}(k-1) + \sum_{q=0}^{N^{(\ell-1)}} w_{jq}^{(\ell)} a_q^{(\ell-1)}(k), & \text{in the case of local activation feedback} \\ w_{jj}^{(\ell)} a_j^{(\ell)}(k-1) + \sum_{q=0}^{N^{(\ell-1)}} w_{jq}^{(\ell)} a_q^{(\ell-1)}(k), & \text{in the case of local output feedback} \end{cases}$$

where the unit's output, $a_j^{(\ell)}(k)$, is a nonlinear function of its linear output, $s_j^{(\ell)}(k)$.

They show that these networks are well suited for applications (such as phoneme classification) in which the network must take relatively recent information into account, yet are poor at tasks for which arbitrarily distant information must be taken into account. Further, they show that the networks which employ local output feedback are capable of information latching; however, they are not capable of classification problems that require counting an arbitrary number of occurrences of the same event.

As mentioned by Frasconi *et al.*, the LRGF with local output feedback resembles Mozer's focused backpropagation network [32]. Mozer refers to units which feed back their output as *context units* (this is commonly used terminology in the neural network literature as the purpose of these feedback connections is to provide temporal context). Mozer proposes an architecture in which the inputs are fed to a layer of context units, the outputs of the context units are fed to a hidden layer of static units, and finally the hidden-layer outputs are fed to the output layer of static units. The main difference between Mozer's context units and the local output feedback units described above is that Mozer keeps the feedback term outside the nonlinearity. For example the output of the j th context unit in the first layer of Mozer's network is given by:

$$a_j^{(1)}(k) = w_{jj}^{(1)} a_j^{(1)}(k-1) + f \left[\sum_{q=0}^{N^{(0)}} w_{jq}^{(1)} a_q^{(0)}(k) \right]$$

as contrasted to the local output feedback units discussed above for which

$$a_j^{(\ell)}(k) = f \left[w_{jj}^{(\ell)} a_j^{(\ell)}(k-1) + \sum_{q=0}^{N^{(\ell-1)}} w_{jq}^{(\ell)} a_q^{(\ell-1)}(k) \right].$$

Mozer claims that inclusion of the feedback term within the nonlinearity causes the context units to forget their values over time.

The Memory Neuron Network discussed below closely resembles Mozer's focused backpropagation network.

5.2.3 Memory neuron networks

The Memory Neuron Network [37] is a simple recurrent network which can be viewed as a feedforward network augmented by recurrent *memory neurons* to provide a certain amount of temporal context. There is one memory neuron associated with each input and with each hidden-layer unit. At each instant of time, the output of a memory neuron is a linear combination of its previous output and the previous output of its associated network unit. Letting $v_j^{(\ell)}(k)$ denote the output of the memory neuron associated with the j th unit (or input in the case $\ell = 0$) on the ℓ th layer, we have that:

$$v_j^{(\ell)}(k) = a_j^{(\ell)} u_j^{(\ell)}(k-1) + (1 - a_j^{(\ell)}) v_j^{(\ell)}(k-1)$$

where $u_j^{(\ell)}(k)$ is the output of the j th unit on layer ℓ at time k and $a_j^{(\ell)}$ is the associated

memory coefficient (an adaptable parameter which assumes values between 0 and 1). The value of the memory coefficient determines the duration of temporal context introduced by the associated memory neuron; coefficients close to 0 result in longer temporal contexts, whereas coefficients close to 1 result in shorter temporal contexts (rapid decay of past information).

The units in the hidden and output layers receive input from each of the units in the previous layer as well as their associated memory neurons. The net input to (or linear output of) the j th unit in layer ℓ is:

$$s_j^{(\ell)}(k) = \sum_{i=1}^{N^{(\ell-1)}} w_{ji}^{(\ell)} u_i^{(\ell-1)}(k) + \sum_{i=1}^{N^{(\ell-1)}} c_{ji}^{(\ell)} v_i^{(\ell-1)}(k) + b_j^{(\ell)}$$

where $w_{ji}^{(\ell)}$ and $c_{ji}^{(\ell)}$ are the connection strengths to unit j on layer ℓ from the i th unit and its associated memory neuron respectively on layer $\ell - 1$, and $b_j^{(\ell)}$ is the bias weight of unit j on layer ℓ . Finally, the output of the j th unit in layer ℓ is a nonlinear function, $f(\cdot)$, of the net input:

$$u_j^{(\ell)}(k) = f(s_j^{(\ell)}(k)) .$$

Poddar and Unnikrishnan [37] have applied the memory neuron network to the prediction of speech waveforms and speech spectra. For the prediction of speech waveforms, they trained two networks, one was trained as a one-step predictor using sampled speech waveforms corresponding to the steady portion of the vowel /a/; the other was trained similarly for the vowel /e/. Each network had one input feeding into two hidden units feeding into one output unit (a total of 14 adaptive parameters). The performance of the two trained networks was compared to the performance of two 20th order Wiener filters (autoregressive (AR) models of the two vowel types). When applied to the vowel type for which the predictor was optimized, the prediction error of the memory network was always less than the prediction error of the AR model. When applied to the opposite vowel type, the prediction error was roughly an order of magnitude greater (for both the AR model and the memory network predictors) than when applied to the correct vowel type.

Poddar and Unnikrishnan have suggested that the prediction networks discussed above could be organized into a larger network to operate as a classifier; a classification decision could be made based on the prediction error from each of the prediction networks.

In a second application, Poddar and Unnikrishnan used the memory network for the prediction of speech spectra (a multivariate time series). In this case, speech signals were preprocessed through a set of 20 filters. At each instant of time, the 20 outputs from the filter bank formed the current input vector to the network. A memory network with 20 inputs, 10 hidden units and 20 output units was trained as a one-step predictor of the speech spectra. The performance of the memory network was compared to the performance of a feed-forward neural network of the same architecture ($20 \times 10 \times 20$). The prediction error of the memory network was on average two orders of magnitude less than the prediction error of the static neural network.

Comparing the true speech spectra to the predicted spectra, Poddar and Unnikrishnan observed that the predicted spectra was a smoothed version of the original spectra but maintained the essential structure; on the other hand the spectra predicted by the static network had a completely different character. They also commented that the prediction error could be used to segment the speech signal since high prediction error could be interpreted as a boundary between different phonetic regions. This idea was previously suggested by Doutriaux and Zipser [8] who found that changes in the activations of the hidden-layer units (of a recurrent network trained as a predictor) could be used to segment speech.

5.3 Fully recurrent networks (Williams-Zipser)

The Williams-Zipser architecture refers to a fully recurrent network in which each unit of the network receives input from the output of all units in the network (including itself) as well as from the external inputs.

Williams and Zipser [55] have applied a fully recurrent network to tasks which require the extraction and storage of information from the input sequence(s) for use in determining the proper output at a later time. Examples of these tasks include: the Pipelined XOR problem (2 external input sequences, 1 output sequence) for which one unit of the network is trained to output at time k the XOR of the two inputs at time $k - \tau$ and a simple sequence recognition task which cannot be solved using a feedforward network with time-

delayed inputs but can easily be solved using a fully recurrent network with only two units (one unit learns to configure itself as a flip-flop and the other as an AND gate). The details of these examples and others can be found in [55].

Kechriotis *et al.* [26] have used a fully recurrent network for adaptive communication channel equalization. They have found that small size fully recurrent network channel equalizers outperform linear transversal filter equalizers in several cases. They admit that the computational complexity of the associated training algorithm is large, but remark that their small size¹⁴ makes them more attractive than feedforward networks (requiring a much larger number of units to model the inverse of the channel).

Although the fully recurrent network offers a very powerful architecture (other recurrent network architectures could be considered as special cases of this one), several authors have noted that the associated training algorithms are slow to converge, and thus, it is often less successful than other less powerful more constrained network architectures (e.g., see section 5.5). In general, placing constraints on the connectivity of the network is equivalent to limiting the solutions that the network might discover; provided that each of several network architectures allow for a satisfactory solution, the training process is likely to be faster for the network architecture which is most constrained.

5.4 Partially recurrent neural networks as NAR and NARMA models

Connor *et al.* [5, 6] have investigated the use of partially recurrent neural networks for nonlinear modeling tasks. Specifically they have compared the performance of two types of recurrent networks to feedforward networks. All networks had a single hidden-layer of nonlinear processing units whose inputs included the past p values of the process to be modeled and whose outputs fed into a single linear output unit (for the case of modeling a one-dimensional process). The networks were trained so as to minimize the squared error between the predicted value of the process (the network's output) and the true value of the process over a training sequence of length 500.

Connor *et al.* [6] refer to the two types of recurrent networks as a *NARMA recurrent network* (RN) and a *NARMA fully recurrent network* (FRN). The NARMA RN feeds back the error between the network's prediction (the network's output) and the true value of

14. Because of the feedback connections, recurrent neural networks with relatively few parameters can be used to equalize nonlinear channels with deep spectral nulls.

the series; the network uses the past q values of this error as additional inputs to the hidden-layer units. The NARMA FRN of Connor *et al.* feeds back the past q outputs of all hidden-layer units as additional inputs to all of the hidden-layer units (they use a network with only one hidden layer). Although the NARMA FRN can realize a richer class of models (the NARMA RN can be viewed as a special case of the NARMA FRN), it has the disadvantage that it may require many more weights than the NARMA RN to realize the same model.

Connor *et al.* [6] evaluated the performance of the various network types on several types of processes including: a gaussian autoregressive (AR) process, a gaussian moving-average (MA) process, and a nonlinear autoregressive moving-average (NARMA) process. Various numbers of hidden-layer nodes as well as various values of p and q were attempted for each of the network-types. Each network was evaluated with respect to its performance on a test sequence of length 10,000. The performance measure was the normalized mean squared error (nMSE); the normalization of the MSE was such that the optimal predictor would result in a nMSE of 1.

The first experiment consisted of training the various network types to model the following gaussian AR process:

$$x(k) = .9x(k-1) + e(k) .$$

Although, with $p=1$, all three network types achieved near optimal performance, the NARMA RN and the feedforward network performed slightly better than the NARMA FRN. As p increased, it was interesting to watch the MSE on the training set drop to less than 1 while the MSE on the test set increased due to poor generalization (or equivalently, memorization of the noise).

Next, the networks were trained to model the following gaussian MA process:

$$x(k) = e(k) + 0.9e(k-1) .$$

As may have been expected, the NARMA RN (with $q=1$ and ($p=0$ or $p=1$)) exhibited the best performance (a test set nMSE of 1.02) for this moving average process; the NARMA FRN (with $q=1$ and $p=1$) did slightly worse (a test set MSE of 1.05) probably due to the fact that the NARMA RN model could realize a close approximation of the desired model with fewer weights; although the feedforward network had the worst performance, its performance improved as p increased (the test set MSE decreased from 1.36

to 1.09 as p increased from one to five), thus illustrating that recurrent networks hold some of the same advantages over feedforward networks as are held by ARMA models over AR models.

In a third experiment, the networks were trained to model the following bilinear NARMA process:

$$x(k) = e(k) + 0.5e(k-1)x(k-1) .$$

In this case, the NARMA FRN (with $p=1$ and $q=1$) achieved the best performance (test set nMSE of 1.11). The NARMA RN (with $p=1$ and $q=1$) had the next best performance (test set nMSE of 1.28). The feedforward network resulted in the worst performance; although its performance improved as p increased from one to two (test set nMSEs of 1.48 for $p=1$ and 1.35 for $p=2$), further increases in p for the feedforward network resulted in significantly worse performance (test set nMSEs close to 1.5 for $p=3,4$, and 5). This experiment illustrated the fact that the NARMA FRN may be better at modelling certain types of nonlinearities than the NARMA RN.

The experiments conducted by Connor *et al.* [6] have demonstrated that although recurrent networks do not have any great advantage over feedforward networks (with time-delayed inputs) for modeling and predicting simple autoregressive processes; they may have considerable advantage over feedforward networks for modeling NARMA processes. As such, they have demonstrated that recurrent neural networks are well-suited for modeling "times series that possess moving average components, are state dependent, or have trends" [5]. They caution that the choice of model is critical to good prediction performance.

5.5 Comparison of LRGF, static feedforward, and fully recurrent neural networks for Speech Prediction

Tsoi and Back [44] have conducted some preliminary experiments in order to compare the performance of LRGF's (both a synaptic-feedback LRGF of section 5.2.1 and an output-feedback LRGF of section 5.2.2) to the performance of a static feedforward network and the Williams-Zipser fully recurrent network (of section 5.3) on a speech prediction task. The training set consisted of a single segment of speech extracted from a nonlinear region of a speech waveform; the speech signal was sampled at 16KHz and the extracted segment was 16ms (256 samples) in duration. All the networks were trained as one-step

predictors for the speech segment. The performance comparison was based on the mean square prediction error and the variance of the prediction error.

The synaptic-feedback LRGF ($1 \times 5 \times 1$) used IIR connections (with $M_b = 5$ and $M_a = 5$ [see (18) for definitions of M_b and M_a]; the output-feedback LRGF ($1 \times 5 \times 1$) used the previous output of each unit as an additional input to the unit (as described in section 5.2.2); the Williams-Zipser network consisted of five fully connected units (one of which was an output unit); and the static feedforward network ($5 \times 5 \times 1$) used five time-delayed inputs.

Each network was trained for 1500 epochs of the selected speech segment (due to poor performance, the Williams-Zipser architecture was trained for an additional 3500 epochs). The resulting mean square prediction error and prediction-error variances for the four network types are summarized in the Table 1 below.

Table 1: Performance Comparison of four network types for a Speech Prediction Task conducted by Tsoi and Back [44].

Architecture	MSE	Variance
LRGF with synaptic feedback	0.0522	0.0097
LRGF with output feedback	0.0225	0.0041
fully recurrent (Williams-Zipser)	0.1803	0.1777
static feedforward network	0.0299	0.0097

From Table 1, we see that the LRGF with output feedback exhibited the best performance; it was closely followed by the static feedforward network. The fully recurrent network had, by far, the worst performance. Tsoi and Back caution that these are preliminary results and that no attempt was made to optimize any of the networks for the task. Furthermore, it should also be noted that these errors and variances are evaluated on the same segment of speech for which the networks were trained. No attempt was made to evaluate the performance of the trained networks on other speech segments.

5.6 Training algorithms for recurrent neural networks

Most of the training algorithms which have been proposed for recurrent neural networks are gradient-descent type algorithms. As such, they require a means for evaluating

the so-called ordered partial derivative (also known as the dynamic derivative) of the cost function with respect to each of the weights of the neural network. As reviewed by Piche [36], there are two basic ways in which the chain rule may be expanded and applied for this purpose. Although the two expansions result in equivalent¹⁵ training algorithms, the algorithms may vary considerably in memory and computational requirements. Piche refers to the two types of algorithms as *backsweep* and *recursive*. The backsweep algorithm is commonly referred to as backpropagation-through-time and the recursive algorithm is often referred to as real-time-recurrent-learning or real-time backpropagation. In general, the backsweep algorithms are less computationally intensive but have higher associated storage costs than the recursive algorithms. The recursive algorithms are more readily adapted to real-time learning.

Other reviews of training algorithms for recurrent neural networks can be found in [3, 38, 34, 33, 55, 52].

15. The training algorithms are equivalent in that they result in the same weight changes provided that epochwise updates are made. Beaufays and Wan [3] have established very nicely the equivalence between the backsweep and recursive type algorithms using flow graph interreciprocity.

6 Classification of Acoustic Transients Using Neural Networks

The classification of short-duration oceanic signals obtained by a passive sonar system has been recognized by many as a challenging problem [43]. The challenges arise from the fact that the signals [13]:

- are highly nonstationary
- originate from a large and diverse group of sources
- contain considerable intraclass variation (due to changes in the oceanic environment, different propagation paths, and multipath effects)
- may overlap with one another, and
- often require event association for proper identification.

As discussed in section 2 of this report, neural networks have several characteristics which deem them worthy of consideration for such a challenging task. Notably: their ability to learn by example obviates a comprehensive characterization of short-duration oceanic signals; their ability to adapt offers the potential of maintaining good performance as the oceanic environment changes; their ability to realize complex mappings allows implementation of the complex decision boundaries required for good classification performance; and their parallel nature offers the opportunity for high-speed processing.

Recently, several research groups have applied neural networks to the task of classifying acoustic transients received by a passive sonar system. The work of three of these groups (Ghosh *et al.* [13], Hemminger and Pao [19], and Kundu *et al.* [28]) is recounted below. Despite the fact that all three groups have used the same recording of acoustics transients, supplied by the Defense Advanced Research Projects Agency (DARPA), it is difficult to compare their results as: the training and test tokens differed from group to group (it appears that each group extracted their own training and test tokens); one group included noise as a possible signal class¹⁶ whereas the other two groups did not; the SNRs may have varied from group to group (this is unclear); and the amount of preprocessing prior to feature extraction varied among the groups.

16. Kundu *et al.* [28] argue that the inclusion of noise as a separate signal class makes the classification task more difficult as the ambient noise has a frequency spectrum that substantially overlaps with the spectra of classes A, B, E, and F.

Below, we present a brief summary of the DARPA data set (as understood from the descriptions provided in [13, 19, 28]) followed by a brief summary of the methods and results of the three research groups.

6.1 Description of the DARPA data set

The DARPA data set consists of two digital time recordings (one for training and one for testing) of six signal types which were propagated over short oceanic data paths. A description of the six signal classes (labelled A through F) is provided below (ocean ambient noise, class *N*, has been included by two of the research groups as a seventh class):

Class <i>A</i> :	Broadband 15 ms pulse
Class <i>B</i> :	Two 4 ms pulses, 27 ms separation
Class <i>C</i> :	3 KHz tonal, 10 ms duration
Class <i>D</i> :	3 KHz tonal, 100 ms duration
Class <i>E</i> :	150 Hz tonal, 1 s duration
Class <i>F</i> :	250 Hz tonal, 8 s duration
Class <i>N</i> :	Ocean ambient noise

The recordings included various levels of source and background noise resulting in a variety of SNRs. Two groups, [13] and [28], indicated that the SNRs varied over a range of 24 dB without stating the specific range of SNRs involved. The third group, [19], mentioned only that the SNRs for class *A* ranged between 1 dB and 7 dB and that the highest SNR from class *B* was 4.3 dB. The size of the training and testing sets varied from one group's description to the next and will thus be discussed separately for the individual groups.

6.2 Description of the DARPA data set by Ghosh, Deuser, and Beck

Ghosh, Deuser, and Beck [13] extracted only those portions of the recordings that contained signals; they did not include ocean ambient noise as a possible class. They report a training set size of 42 examples with the number of examples per class ranging between 5 and 9. They divided the patterns extracted from the test recording into two test sets: the first set included 179 examples which were similar to the training data (we will refer to this set as the *similar test set*); the second set consisted of an additional 19 examples with frequency characteristics which deviated significantly from the training data (we will refer to this set as the *deviant test set*). They used the deviant test set to evaluate the robustness

of their techniques. The composition of their training and testing sets are summarized in Table 2 below.

Ghosh *et al.* [13] report an SNR range of 24 dB but did not indicate the specific range of SNRs which were included.

Table 2: Composition of Training and Test Sets Used by Ghosh, Deuser, and Beck

Class	Number of samples of specified class in specified set		
	training set	"similar" test set	"deviant" test set
A	7	53	8
B	7	54	6
C	8	31	1
D	9	14	0
E	6	19	4
F	5	8	0
TOTALS	42	179	19

6.2.1 Preprocessing and feature extraction

Ghosh *et al.* [13] "use an adaptive time-domain whitening filter to decorrelate the transients from the long-time ambient noise, interference, and sensor characteristics, while passing short-duration signals relatively unchanged." They also indicate that the data is normalized but do not provide details.

In a previous study, they found that wavelet-based features yielded better performance than AR parameters or spectral coefficients; hence, they have used a constant-Q prototype wavelet with 24 coefficients for feature extraction. In addition to the 24 wavelet-based coefficients, they use the time-duration of the signal as an added classification parameter. They make no mention of how they extract the time duration of the signal nor of how much variation there was in the time durations of each class. Thus, unfortunately, the extent to which the inclusion of the time duration simplifies the problem is unclear (especially since each signal class is characterized by a unique time duration and, in fact the time duration is the only discriminant between classes C and D); it would be useful to know the classification accuracy which could be achieved with just the single time-duration feature.

6.2.2 Classifiers attempted and summary of classification results

Ghosh *et al.* [13] evaluated the performance of six types of classifiers including: a 3-nearest neighbor (3NN) classifier, a radial basis function (RBF) network, a Learning Vector Quantization 2 (LVQ2) network, an adaptive kernel classifier (AKC) network, a rapid kernel classifier (RKC) network, and a Pi-Sigma network (PSN). The RBF network used a total of 15 centroids; the LVQ2, the AKC, and the RKC all used 30 reference vectors (5 reference vectors for each class); the PSN used 3 hidden units. Descriptions of K-NN classifiers, RBF networks, and LVQ2 can be found in [10], [20], and [27] respectively; the AKC, RKC, and PSN networks are all described by Ghosh *et al.* [13]. The operation of each of these classifiers is static. The classification rate achieved by each of the six classifiers on the two test sets described in section 6.2 is summarized in the table below.

As can be seen from the table, each of the classifiers is able to correctly classify close to 100% of the patterns in the "similar" test set, yet a relatively low percentage of the patterns in the "deviant" test set are correctly classified. Confusion matrices for the patterns in the two test sets are supplied in [13]. From the confusion matrices, it is clear that the patterns which are incorrectly classified vary from classifier to classifier.

Table 3: Classification Results of Ghosh, Deuser, and Beck

Classifier-Type	Classification Accuracy (percent)	
	"similar" test set	"deviant" test set
3-NN	96.1	21.1
RBF	98.9	21.1
LVQ2	97.2	26.3
AKC	100	21.1
RKC	98.9	21.1
PSN	98.9	31.6
synergistic approach # 1 RKC & PSN	100	42
synergistic approach # 2 RKC & PSN	100	47

6.2.3 Synergistic approach to classification (Ghosh *et al.*)

Ghosh *et al.* [13] suggest that a synergistic approach to the classification task may yield better solutions than could be provided by making further improvements to any single ap-

proach. With this in mind, they propose two different methods for combining the classification decisions of n different classifiers to yield a more robust classification decision. Using their two proposed approaches to combine the decisions of the RKC and PSN classifiers improves the classification accuracy on the deviant test set to 42% and 47% as shown in the final two rows of Table 3, while maintaining a classification accuracy of 100% on the similar test set.

6.3 Description of the DARPA data set by Kundu, Chen, and Persons

Kundu, Chen, and Persons [28] describe the DARPA data set as consisting of the seven signal classes enumerated in section 6.1 (they include noise as a separate signal class and state that the inclusion of noise makes the classification task more difficult as the ambient noise has a frequency spectrum that substantially overlaps with the frequency spectra of classes A, B, E, and F). In contrast to Ghosh *et al.*, they report a training set size of 161 patterns (23 patterns for each signal class) and a test set size of 154 patterns (22 patterns from each signal class). Their training and testing sets include patterns which are representative of five different SNR groups, the lowest SNR being 24 dB less than the highest.

6.3.1 Preprocessing and feature extraction

Unlike the other two groups, Kundu *et al.* [28] did not attempt to remove the effects of background noise prior to feature extraction.

Feature extraction was performed on each of the extracted signal segments. Each extracted segment consisted of 1024 data points (corresponding to a duration of approximately 42 ms at the specified sampling rate of 24.576 KHz). Prior to feature extraction, the 1024 data points were subdivided into 21 overlapped time frames of 256 data points; the overlap between consecutive time frames was 218 points or approximately 85%. Feature vectors were computed for each time frame. Thus, the resulting patterns consisted of a sequence of 21 feature vectors.

They experimented with several different types of feature vectors. These included: a feature vector consisting of sixth-order AR coefficients, a feature vector consisting of

tenth-order AR coefficients, a feature vector consisting of a selected¹⁷ group of 30 FFT power-spectral coefficients, a feature vector consisting of a selected¹⁷ group of 30 Daubechies-4 wavelet coefficients, and a feature vector consisting of a selected¹⁷ group of 30 Daubechies-20 wavelet coefficients.

6.3.2 Classifiers attempted and summary of classification results

Kundu *et al.* [28] report experimental results for three types of classifiers: a 6-state Hidden Markov Model (HMM); an 8-state HMM; and a multilayer perceptron¹⁸ (MLP) with 20 hidden-layer nodes. For presentation to the MLP, the sequence of feature vectors were concatenated and presented simultaneously.

The classification results obtained by Kundu *et al.* [28], for the patterns in their test set, are summarized in Table 4 below. From the table, we see that the AR coefficients were considerably less useful for classifying the acoustic transients than either the FFT coefficients or the wavelet-based coefficients. The best results were obtained with the Daubechies-20 coefficients and the MLP.

Table 4: Classification Results of Kundu, Chen, and Persons

Classification Accuracy for Selected Combinations of Classifiers and Feature Vectors			
Feature-Vector Type	Classifier Type		
	6-state HMM	8-state HMM	MLP
6th order AR coefficients	73.3%	worse than 6-state HMM	—
10th order AR coefficients	67.5%	—	—
30 FFT coefficients	—	89.6%	90.9%
30 Daubechies-4 coefficients	—	91.5%	90.9%
30 Daubechies-20 coefficients	—	90.9%	93.5%

6.3.3 Synergistic approach to classification (Kundu *et al.*)

Similar to Ghosh *et al.* [13], Kundu *et al.* [28] suggested the use of a combined classifier. Their approach was to make a decision based on a majority vote (above some threshold)

17. The selection process consisted of identifying a set of significant coefficients (those coefficients with large magnitudes) for each pattern class; the coefficients in the union of these sets were retained as features.

18. The multilayer perceptron is synonymous with a multilayer feedforward neural network.

of the six classification systems defined by the six possible feature-vector/classifier combinations which arise from a choice of three types of feature vectors (the selected FFT coefficients, the selected Daubechies-4 coefficients, and the selected Daubechies-20 coefficients) and two types of classifier (the MLP and the 8-state HMM). When using a threshold of 4 (*i.e.*, at least 4 of the systems had to agree in order for a decision to be made) only two patterns were misclassified and 12 patterns were left unclassified; thus 90.9% of all the test patterns were correctly classified, and of the patterns for which a decision was made, 98.6% of the decisions were made correctly. When using a threshold of three, five patterns were misclassified and three patterns were left unclassified; in this case, 94.8% of all the test patterns were correctly classified, and of the patterns for which a decision was made, 96.7% of the decisions were made correctly.

6.4 Description of the DARPA data set by Hemminger and Pao

Similar to Ghosh *et al.* [13], Hemminger and Pao [19] did not include noise as a possible signal class; they extracted only those portions of the recordings that contained signals. They do not explicitly mention the size of their training set; however, based on details provided in various figure captions, it appears that their training set included seven examples from class A, four examples from class B, and eight examples from class C; they never mention the number of training examples from classes D, E, and F. They report results for a test set which includes a total of 218 patterns (72 from class A, 58 for class B, 40 for class C, 12 for class D, 21 for class E, and 15 for class F).

Rather than stating that the SNRs varied over a range of 24 dB as did the other two research groups, Hemminger and Pao simply mentioned that a variety of SNRs were included. They also provided two specific pieces of information not mentioned by the other two groups: namely that their class A signals had SNRs between 1 dB and 7 dB and that the highest SNR for class B signals was 4.3 dB (they do not state the lowest SNR for type B signals). No other details on SNRs were provided except in their conclusions where they state: "In general, the average class SNR can drop to about 0.5 dB and still be successfully classified." As their classification results were very good, it seems logical to conclude that the average SNRs were greater than or equal to 0.5 dB.

6.4.1 Preprocessing and feature extraction

The preprocessing used by Hemminger and Pao [19] consisted of computing the power spectra associated with each record and then, in the case of a noisy environment, subtracting off a moving average of the background noise power spectra. To compute the power spectra, each signal was broken down into several overlapping records (75% overlap) of 256 points (corresponding to a record length of approximately 10 ms for their reported sampling rate of 25KHz). Rather than performing a single STFT on each record of 256 points, they further subdivided each record into seven overlapping segments of 64 points (50% overlap), computed the STFT for each 64 point segment and then averaged the seven spectra to get the spectrum representative of the 256 point record. They then subtracted off a moving average of the background noise spectra and normalized the resulting spectra by the RMS value of its components.

Feature extraction required knowledge of the signal characteristics. This knowledge was extracted from the training signals in two stages. In the first stage, a clustering algorithm was applied to the spectra computed for all records in the training set to determine a representative group of prototypical spectra. The objective of the second stage was to characterize each signal class with a single sequence of spectra prototypes. The length of the characterizing sequence varied from class to class depending on the duration of typical signals in each class. The characterizing sequence length was 5 for class A, 17 for class B, 5 for class C, and 13 for classes D, E, and F.

Several feature vectors were extracted from each signal (one feature vector was extracted for each possible signal class). The Class A feature vector, extracted from a given input signal, was the same length as the characterizing sequence for Class A and consisted of the modified Hausdorff distances between corresponding spectra in the signal's spectra sequence and the class A characterizing sequence. The feature vectors for the other classes were defined similarly.

6.4.2 Classifiers attempted and summary of classification results

The classifier consisted of several neural networks (one for each possible signal class). Each neural network consisted of a single linear unit. The inputs to the network associated with class c , consisted of the features (the Hausdorff distances) in the associated class c feature vector as well as the second-order cross terms of these features. Each network

is trained to recognize Hausdorff distances typical of signals from the associated signal class.

The resulting classification system was tested with both signals and noise. During testing, a signal was classified according to the network having the output which most exceeded its predetermined threshold. If none of the networks' output exceeded its predetermined threshold, the signal was not associated with any class. Out of their 218 test signals, none was classified wrongly, although nine of them were left unclassified. Hemminger and Pao [19] stated that "in general, the average class SNR can drop to about 0.5 dB and still not be a problem; they also stated that noise had never triggered any false alarms (*i.e.*, that noise had never caused any of the networks' output to rise above its predetermined threshold).

6.5 Application of dynamic neural networks to acoustic transient classification

With the possible exception of Ghosh *et al.*¹⁹, all previously-mentioned groups chose to represent the oceanic signals to-be-classified by a feature vector which consisted of sequential sets of features (one set for each of several overlapping segments of the signal). However, rather than presenting these feature sets sequentially to a dynamic neural network classifier, each group concatenated the sequence of feature sets into a single feature vector which was then presented to a static network for classification.

Although Ghosh *et al.* state that dynamic classifiers are more powerful and promising for classifying complex temporal patterns, they did not present any results for such an approach²⁰. In fact, we are not aware of any published work which makes use of a dynamic neural network for classification of acoustic transients received by a passive sonar system. This may be due to the following facts: in comparison to static neural networks, dynamic neural networks are relatively novel and their properties are not as well understood; simulations of dynamic neural networks are not as readily available as they are for static networks; and recordings of acoustic transients available to the public are scarce.

19. It was not clear whether the wavelet coefficients extracted by Ghosh *et al.* represented a sequence of time windows or a single window of time; they seem to hint at the latter.

20. They did however claim to be currently investigating such an approach.

Chapters 4 and 5 recounted numerous successful applications of dynamic neural networks to speech processing tasks. Several similarities exist between the phoneme-classification task and the acoustic transient classification task, namely:

- Both tasks involve the classification of acoustic waveforms that evolve in time.
- Both tasks are complicated by the fact that it is virtually impossible to define a "typical" pattern associated with each class (*i.e.*, there is significant intraclass variation).
- Both tasks are complicated by anticipated time misalignments due to inconsistent segmentation.

Based on the above-mentioned similarities between the phoneme-classification task and the acoustic transient classification task, it seems logical to conclude that dynamic neural networks may offer several advantages over static neural networks, for classification of acoustic transients.

In an attempt to demonstrate the potential capabilities of dynamic neural networks, we have applied a FIR neural network to several simulated data sets. The first data set contained considerable intraclass variation. The second simulated data set was based somewhat on the descriptions of the DARPA data set in section 6.1 of this report. The third data set consisted of signals received via a simulated multipath channel (the model of the multipath channel incorporated several elements of randomness). We have successfully applied a FIR network to all three data sets. For two of the data sets, we have compared the performance of the FIR network to that of the static network; in both cases, the performance of the FIR network was superior or comparable to that of the static network. Our results are summarized in [1].

7 Summary and Conclusions

This report has reviewed dynamic neural networks and their applications to temporal processing tasks. The organization of the report was discussed in chapter 1 and background material was provided in chapter 2.

Special emphasis has been placed on a very simple dynamic feedforward neural network known as the Finite Impulse Response Neural Network (FIRNN) which was introduced in Chapter 3. It was shown that any solution which can be implemented by an FIRNN can also be implemented by a static network operating on a spatially expanded representation of the original spatio-temporal input pattern. The equivalent static network implementation will have many more associated weights than the FIRNN implementation; however, many of the static network's weights will have the same value, and in fact, the number of distinct weight values in the equivalent static neural network implementation will be identical to the number of distinct weight values in the FIRNN implementation. For temporal processing tasks, the FIRNN has a distinct advantage over the static neural network in that its architecture imposes an appropriate type of solution: namely a solution for which units in the first layer operate as time-invariant detectors of relatively short-duration events, while the units in succeeding layers function to detect events of longer duration (possibly concatenations of shorter-duration events). To find a similar solution using a static neural network would be more difficult as the static network would have to learn how to configure itself as an FIR network (*i.e.*, it would have to learn the various constraints among the weights) in addition to finding appropriate values for the weights. The FIRNN has been very successfully applied to both prediction and classification tasks as evidenced in chapter 4.

Other types of dynamic networks were introduced in chapter 5. These included other types of dynamic feedforward networks as well as recurrent neural networks. The dynamic feedforward networks which were discussed included the Time Derivative Neural Network and the Tempo 2 Network. The Time Derivative Neural Network offers the advantage of increased temporal context without an associated increase in the number of weights. The Tempo 2 network allows for the automatic adjustment of the temporal context for each connection between the input and hidden-layer units. This is important as the process of selecting a suitable temporal context for a task can be very laborious. Both networks have

been compared to the FIRNN for certain temporal-processing tasks and have demonstrated comparable or superior performance.

Recurrent neural networks are characterized by the fact that they incorporate some sort of feedback. As such they offer advantages over dynamic feedforward networks similar to the advantages offered by ARMA models over AR models and the advantages offered by IIR filters over FIR filters; namely they are able to model resonances and sharp cutoffs more parsimoniously. A parsimonious model is desirable as it typically generalizes better than one with a large number of parameters which allows noise to be modelled in addition to the signal of interest.

The standard recurrent neural networks introduced in chapter 5 were categorized (according to feedback-type) as locally recurrent globally feedforward (LRGF), partially recurrent, and fully recurrent. Although the fully recurrent network is the most powerful, it typically requires much longer training times. The LRGF and partially recurrent networks can be considered as special constrained cases of the fully recurrent network. In general, placing constraints on the connectivity of the network can be viewed as limiting the solutions that the network might discover; provided that each of several network architectures can realize the sought solution, the training process is likely to be faster and more successful for the network architecture which is most constrained. Unfortunately, determination of the most appropriate architecture for a particular problem is difficult and requires either a thorough understanding of the problem, extensive experimentation, or both.

Applications of the various types of recurrent networks were also presented in chapter 5. The application areas included prediction and classification of speech waveforms using static feedforward, dynamic feedforward, LRGF, and fully recurrent neural networks; modelling of NAR and NARMA processes using partially recurrent networks; and equalization of nonlinear communication channel using fully recurrent networks.

Chapter 6 discussed the difficulties inherent to acoustic transient processing and recounted a variety of techniques attempted by three research groups who applied neural networks to the classification of acoustic transients on a recording supplied by the Defense Advanced Research Projects Agency (DARPA); the acoustic transients on the DARPA supplied recording had been propagated over short duration oceanic paths, with variable

source and background noise levels, and received by a passive sonar system. The three groups applied various noise-removal, preprocessing, feature-extraction, and classification techniques to the recorded transients. Each group appeared to be somewhat successful; their results are difficult to compare for reasons discussed in chapter 6. Two of the groups recommended a synergistic approach to classification of acoustic transients in which the outputs of two or more classifiers are combined to yield a more robust classification decision. At least one group acknowledged the fact that the DARPA supplied data set was a simplified characterization of real-life short-duration oceanic signals.

Although there is very little published work available on the application of dynamic neural networks to the classification of acoustic transients (the research groups mentioned in the preceding paragraph had all used static neural networks as classifiers), it is the author's opinion that dynamic neural networks have much to offer. This opinion is based, in part, on the parallels which exist between speech recognition and acoustic transient classification; the initial successes of dynamic neural networks applied to problems in speech recognition, as well as our own preliminary experimental results discussed in [1].

REFERENCES

- [1] Arsenault, N. and M. Stevenson, *Classification of Simulated Sequences using an FIR Neural Network: Experimental Results*, CR/97/459, April 1995.
- [2] Back, A.D. and A.C. Tsoi, "FIR and IIR Synapses, a New Neural Network Architecture for Time Series Modeling," *Neural Computation*, vol. 3, pp. 375-385, 1991.
- [3] Beaufays, F. and E.A. Wan, "Relating Real-Time Backpropagation and Backpropagation-Through-Time: An Application of Flow Graph Interreciprocity," *Neural Computation*, vol. 6, pp. 296-306, 1994.
- [4] Bodenhausen, U. and A. Waibel, "The Tempo 2 Algorithm: Adjusting Time-Delays By Supervised Learning," In *Advances in Neural Information Processing Systems*, eds. R.P. Lippmann, J.E. Moody, and D.S. Touretzky, vol. 3, pp. 155-161, Morgan Kaufmann Publishers, Inc., 1991.
- [5] Connor, J., L.E. Atlas, and D.R. Martin, "Recurrent Networks and NARMA Modeling," In *Advances in Neural Information Processing Systems*, eds. J.E. Moody, S.J. Hanson, and R.P. Lippmann, vol. 4, pp. 301-308, Morgan Kaufmann Publishers, Inc., 1992.
- [6] Connor, J.T., R.D. Martin, and L.E. Atlas, "Recurrent Neural Networks and Robust Time Series Prediction," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240-254, March, 1994.
- [7] Cybenko, G., "Approximation by superpositions of a sigmoidal function," *Math of Control, Signals, and Systems*, vol. 2, pp. 303-314, 1989.
- [8] Doutriaux, A. and D. Zipser, "Unsupervised Discovery of Speech Segments using Recurrent Networks," *Connectionist Models Proceedings of the 1990 Summer School*, edited by: D.S. Touretzky, J.L. Elman, T.J. Sejnowski, G.E. Hinton, Morgan Kaufmann Publishers, 1991, pp. 303-309.
- [9] Draper, N.R. and H. Smith, *Applied Regression Analysis*, Wiley, New York, 1981.
- [10] Duda, R.O. and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
- [11] Elman, J.L., "Finding Structure in Time," *Cognitive Science*, vol. 14, pp. 179-211, 1990.
- [12] Frasconi, P., M. Gori, and G. Soda, "Local Feedback Multilayered Networks," *Neural Computation*, vol. 4, pp. 120-130, 1992.
- [13] Ghosh, J., L.M. Deuser, and S.D. Beck, "A Neural Network Based Hybrid System for Detection, Characterization, and Classification of Short-Duration Oceanic Signals," *IEEE Journal of Oceanic Engineering*, vol. 17, no. 4, October, 1992.
- [14] Gori, M., "An Extension of BPS," In *Proceedings of Neuro-Nimes '89* (Nimes, France), pp. 83-93, November, 1989.

-
- [15] Gorman, R. P. and T. J. Sejnowski, "Learned classification of sonar targets using a massively parallel network," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 7, pp. 1135-1140, July 1988.
- [16] Gorman, R. P. and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, vol. 1, no. 1, pp. 75-89, July 1988.
- [17] Haykin, S., *Neural Networks A Comprehensive Foundation*, Maxwell Macmillan Canada, Inc., Don Mills, Ontario, 1994.
- [18] Hecht-Nielsen, R., "Theory of the backpropagation neural network," In *Proceedings of the International Joint Conference on Neural Networks*, 1989, June, vol. 1, pp. 593-611.
- [19] Hemminger, T.L. and Y.H. Pao, "Detection and Classification of Underwater Acoustic Transients Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 5, September, 1994.
- [20] Hertz, J., A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, Redwood City, California, 1991.
- [21] Hinton, G.E., and T.J. Sejnowski, "Learning and Relearning in Boltzman Machines," D.E. Rumelhart, J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*, Chapter 7, Cambridge, MA, MIT Press, 1986.
- [22] Hopfield, J.J., "Neural Networks and Physical Systems with emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554-2558, 1982.
- [23] Hopfield, J. and N. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, vol. 52, pp. 141-152, 1985.
- [24] Hornik, K., M. Stinchcombe, and H. White, "Multi-layer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359-366, 1989.
- [25] Hush, D.R. and B.G. Horne, "Progress in Supervised Neural Networks. What's New Since Lippmann?" *IEEE Signal Processing Magazine*, pp. 8-39, January, 1993.
- [26] Kechriotis, G., E. Zervas, and E.S. Manolakos, "Using Recurrent Neural Networks for Adaptive Communication Channel Equalization," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, March, 1994.
- [27] Kohonen, T., *Self-Organization and Associative Memory (3rd edition)*, Springer-Verlag, Berlin, 1989.
- [28] Kundu, A., G.C. Chen, and C.E. Persons, "Transient Sonar Signal Classification Using Hidden Markov Models and Neural Nets," *IEEE Journal of Oceanic Engineering*, vol. 19, no. 1, January, 1994.
- [29] Lippmann, R. P., "An Introduction to Computing with Nerual Nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

-
- [30] Lippmann, R.P., "Review of Neural Networks for Speech Recognition," *Neural Computation*, vol. 1, no. 1, 1989.
- [31] McCulloch, W.S. and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.
- [32] Mozer, M., "A Focused Backpropagation Algorithm for Temporal Pattern Recognition," *Complex Systems*, vol. 3, pp. 349-381, 1989.
- [33] Nerrand, O., P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos "Neural Networks and Nonlinear Adaptive Filtering: Unifying Concepts and New Algorithms," *Neural Computation*, vol. 5, pp. 165-199, 1993.
- [34] Nerrand, O., P. Roussel-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus, "Training Recurrent Neural Networks: Why and How? An Illustration in Dynamical Process Modeling," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 178-184, 1994.
- [35] Paliwal, K.K., "A Time-Derivative Neural Net Architecture - An Alternative to the Time-Delay Neural Net Architecture," In *Neural Networks for Signal Processing - Proceedings of the 1991 IEEE Workshop*, eds. B.H. Juang, S.Y. Kung, and Candace A. Kamm, pp. 367-375.
- [36] Piche, S.W., "Steepest Descent Algorithms for Neural Network Controllers and Filters," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 198-212, March, 1994.
- [37] Poddar, P. and K.P. Unnikrishnan, "Non-linear Prediction of Speech Signals using Memory Neuron Networks," In *Neural Networks for Signal Processing - Proceedings of the 1991 IEEE Workshop*, eds. B.H. Juang, S.Y. Kung, and Candace A. Kamm, pp. 395-404.
- [38] Puskorius, G.V. and L.A. Feldkamp, "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 279-297, March, 1994.
- [39] Rabiner, L.R. and B.H. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine*, pp. 4-16, June, 1986.
- [40] Rioul, O. and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine*, pp. 14-38, October, 1991.
- [41] Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 8, MIT Press, Cambridge, MA, 1986.
- [42] Silven, S., "A Neural Approach to the Assignment Algorithm for Multiple-Target Tracking," *IEEE Journal of Oceanic Engineering*, vol. 17, no. 4, October, 1992.
- [43] Special Issue, "Underwater Acoustic Signal Processing," *IEEE Journal of Oceanic Engineering*, pp. 2-278, January, 1987.

-
- [44] Tsoi, A.C. and A.D. Back, "Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 229-239, March, 1994.
- [45] Waibel, A., T. Hanazawa, G. Hinton, K. Shikano, and K. Lang "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328-339, 1989.
- [46] Waibel, A., "Modular construction of time-delay neural networks for speech recognition," *Neural Computation*, vol. 1, no. 1, pp. 39-46, 1989.
- [47] Wan, E.A., *Finite Impulse Response Neural Networks with Applications in Time Series Prediction*, Ph.D. dissertation, Electrical Engineering Dept., Stanford University, November, 1993.
- [48] Wan, E.A., "Temporal backpropagation: an efficient algorithm for finite impulse response neural networks," In *Proceedings of the 1990 Connectionist Models Summer School*, Morgan Kaufmann, pp. 131-140.
- [49] Wan, E.A., "Temporal backpropagation for FIR neural networks," In *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, San Diego, pp. 575-580, 1990.
- [50] Weigend, A.S., B.A. Huberman, and D.E. Rumelhart, "Predicting the Future: A Connectionist Approach," *International Journal of Neural Systems*, vol. 1, no. 3, pp. 193-209, 1990.
- [51] Weigend, A.S. and N. Gershenfeld, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison-Wesley, 1993.
- [52] Werbos, P.J. "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550-1560, 1990.
- [53] Widrow, B. and S.D. Stearns, *Adaptive Signal Processing*, Prentice Hall, Inc., Englewood Cliffs, N.J., 1985.
- [54] Widrow, B., D.E. Rumelhart, and M.A. Lehr, "Neural Networks: Applications in Industry, Business, and Science," *Comm. of the ACM*, vol. 37, no. 3, pp. 93-105, March, 1994.
- [55] Williams, R.J., and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. 1, pp. 270-280, 1989.

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA		
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)		
1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.) University of New Brunswick P.O. Box 440, Fredricton, N.B., Canada E3B 5A3		2. SECURITY CLASSIFICATION (Overall security of the document including special warning terms if applicable.) UNCLASSIFIED
3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C, R or U) in parentheses after the title.) Survey of Dynamic Neural Network Techniques with Application to Temporal Processing Tasks		
4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.) STEVENSON, Maryhelen		
5. DATE OF PUBLICATION (Month and year of publication of document.) April 1995	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 70	6b. NO. OF REFS. (Total cited in document.) 55
7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. Interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.) Contractor Report		
8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development. Include the address.) Defence Research Establishment Atlantic P.O. Box 1012, Dartmouth, N.S., Canada B2Y 3Z7		
9a. PROJECT OR GRANT NUMBER (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 1cc	9b. CONTRACT NUMBER (If appropriate, the applicable number under which the document was written.) W7707-3-2682/01-OSC	
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NUMBERS (Any other numbers which may be assigned this document either by the originator or by the sponsor.) DREA Contractor Report 97/458	
11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification) ☒ Unlimited distribution ☐ Distribution limited to defence departments and defence contractors; further distribution only as approved ☐ Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved ☐ Distribution limited to government departments and agencies; further distribution only as approved ☐ Distribution limited to defence departments; further distribution only as approved ☐ Other (please specify):		
12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)		

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

DDOQ 2/86/7

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM

ABSTRACT. To be used and placed at the beginning of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.

Dynamic neural networks and their applications to temporal processing tasks are reviewed. Of special interest is a simple dynamic feedforward network known as the Finite Impulse Response Neural Network (FIRNN) and the classification of temporal patterns such as acoustic transients. Basic principles of the FIRNN are presented along with an explanation of how the FIRNN processes temporal information. Other dynamic networks which are discussed include additional dynamic feedforward neural networks, locally-recurrent globally-feed forward neural networks, and fully recurrent neural networks. Application areas to which the various neural network paradigms have been applied include the classification of acoustic transients, prediction of chaotic time series, prediction and classification of speech signals, and modelling of nonlinear autoregressive (NAR) and nonlinear autoregressive moving-average (NARMA) processes.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloging the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

passive sonar
neural networks
dynamic neural networks
finite impulse response
transient signals

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM

